

BY POPULAR DEMAND XML-J IS NOW MONTHLY!

XML JOURNAL

The World's Leading XML Resource

Volume: 1 Issue: 5

XML-JOURNAL.COM

**XML
DevCon
FALL
2000**

November 12-15, 2000

Announcing...

December 3-5, 2000

**Wireless
DevCon**

FROM THE EDITOR

Reincarnation of Technologies
by Israel Hilerio pg. 5

XML IN TRANSIT

SOAP Part 2
by Simeon Simeonov pg. 14

BUILDING JAVA BUSINESS OBJECTS

**Using Database
MetaData and XSL**
by Brian D. Eubanks pg. 24

PRODUCT REVIEW

FileMaker Developer 5
by FileMaker Inc.
Reviewed by Jim Milbery pg. 52

SYS-CON RADIO

Interview with Sandra Clark
of Insight Technologies
Interviewed by Ajit Sagar pg. 62

XML NEWS

pg. 66

RETAILERS PLEASE DISPLAY
UNTIL DECEMBER 31, 2000
\$4.99US \$6.99CAN

**SYS-CON
MEDIA**

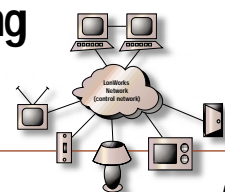
OPAQUE
BODIES,
TRANSPARENT
ENVELOPES

Check out the design
benefit of multilayered
Header/Body
structures

by Joshua Fox
see page 56

XML Feature: Control Appliances Using WML & Java Servlets

Access intelligent appliances via mobile gadgets anywhere, anytime



Siet-Leng Lai

6

XML Feature: A Quick Guide to WML

Develop applications and services for wireless devices

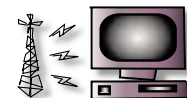


Mahesh Chuleet

20

XML Filter: An Approach to Efficient Information Transfer

Loose coupling protocol provides ease in code change

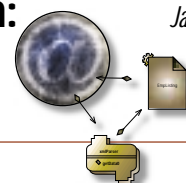


Sameer Bhatia

28

XML Feature: Flexible XML Web Design: XML Data Islands

Combining server-side scripting languages with the power of XML and XSL

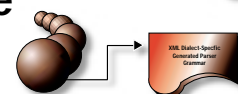


James A. Brannan

34

XML Parsing: High Performance XML Parsing in C++

Parse XML data into C++ classes for increased performance



Ken Blackwell

42

XML Feature: Voice Interaction for the Web

Speech recognition applications are making major breakthroughs



Rick Parfitt

46

SoftQuad Software

[www.softquad.com/
products/xdmetal/eval/](http://www.softquad.com/products/xdmetal/eval/)

Progress

www.sonicmq.com/ad11.htm

Icon Information Systems

www.xmlspy.com



EDITORIAL ADVISORY BOARD

COCO JAENICKE, SIMON PHIPPS, RICK ROSS, AJIT SAGAR, BOB SUTOR

EDITOR-IN-CHIEF: AJIT SAGAR
EXECUTIVE EDITOR: M'LOU PINKHAM
ART DIRECTOR: ALEX BOTERO
MANAGING EDITOR: CHERYL VAN SISE
EDITOR/COPY CHIEF: NANCY VALENTINE
ASSOCIATE EDITOR: JAMIE MATUSOW
E-BUSINESS EDITOR: ISRAEL HILERIO
JAVA TECHNOLOGY EDITOR: JASON WESTRA

WRITERS IN THIS ISSUE

SAMEER BHATIA, KEN BLACKWELL, JAMES A. BRANNAN,
MAHESH CHULET, BRIAN EUBANKS, JOSHUA FOX, ISRAEL HILERIO,
JIM MILBERY, RICK PARFITT, SIMEON SIMEONOV, SIET-LENG LAI

SUBSCRIPTIONS

FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,
PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT

SUBSCRIPTION HOTLINE

800 513-7111

COVER PRICE: \$8.99/ISSUE

DOMESTIC: \$79/YR (12 ISSUES) CANADA/MEXICO: \$99/YR

ALL OTHER COUNTRIES \$129/YR

(U.S. BANKS OR MONEY ORDERS)

PUBLISHER, PRESIDENT AND CEO: FUAT A. KIRCAALI

VICE PRESIDENT, PRODUCTION: JIM MORGAN

VICE PRESIDENT, MARKETING: CARMEN GONZALEZ

GROUP PUBLISHER: LISE ST. AMANT

COMPTROLLER: BRUCE N. MILLER

ADVERTISING ACCOUNT MANAGERS: MEGAN RING

ROBYN FORMA

JDSTORE.COM: AMANDA MOSKOWITZ

SYS-CON EVENTS MANAGER: ANTHONY D. SPITZER

ADVERTISING ASSISTANT: CHRISTINE RUSSELL

ASSOCIATE SALES MANAGER: CARRIE GEBERT

GRAPHIC DESIGNERS: ABRAHAM ADDO

CATHRYN BURAK

GRAPHIC DESIGN INTERNS: AARATHI VENKATARAMAN

LOUIS F. CUFFARI

WEBMASTER: ROBERT DIAMOND

WEB DESIGNERS: STEPHEN KILMURRAY

GINA ALAYAN

CUSTOMER SERVICE: ELLEN MOSKOWITZ

EDITORIAL OFFICES

SYS-CON PUBLICATIONS, INC.

135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645

TELEPHONE: 201 802-3000 FAX: 201 782-9637

SUBSCRIBE@SYS-CON.COM

XML JOURNAL (ISSN# PENDING)

is published monthly (12 times a year) by

SYS-CON Publications, Inc., 135 Chestnut Ridge Road,
Montvale, NJ 07645

3rd Class Postage rates are paid at
Montvale, NJ 07645 and additional mailing offices.

POSTMASTER: Send address changes to:

XML JOURNAL, SYS-CON Publications, Inc.,
135 Chestnut Ridge Road, Montvale, NJ 07645.

©COPYRIGHT

Copyright © 2000 by SYS-CON Publications, Inc. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system, without written permission. For promotional reprints, contact reprint coordinator. SYS-CON Publications, Inc., reserves the right to revise, republish and authorize its readers to use the articles submitted for publication.

WORLD DISTRIBUTION

CURTIS CIRCULATION COMPANY

730 RIVER ROAD, NEW MILFORD, NJ 07646-3048 PHONE: 201 634-7400

All brand and product names used on these pages are trade names,
service marks or trademarks of their respective companies.
SYS-CON Publications, Inc., is not affiliated with the companies
or products covered in XML Journal.



WRITTEN BY ISRAEL HILERIO GUEST EDITOR

from
the
editor

Reincarnation of Technologies

This month I'm sitting in Ajit Sagar's seat while he takes care of family matters associated with the recent death of his father.

As I sat at home thinking about various ways to welcome you to this new edition of **XML-Journal**, I thought there might be a parallel between the loss of a loved one and the death of technologies. Rather than concentrating on the loss, however, I believe it's better to concentrate on the reincarnation of technology. My friend Ajit believes the soul lives on and returns to the world in different forms, and it occurred to me that technologies like XML evolve from legacy technologies following the same process of reincarnation. Let's take a moment to reflect on the technological predecessors of XML and its evolution.

SGML (Standard Generalized Markup Language) was one of the technologies that gave birth to XML. The Department of Defense, in their Interactive Electronic Technical Manuals (IETM), adopted SGML as their markup language for such manuals. XML is a subset of SGML; thus all XML documents conform to SGML documents. The idea of markup is to encode a description of a document storage layout and logical structure. One of the ideas behind storage units and logical structure is its ability to capture semantic information. The tag name and its place in the structure identify the description of value contained inside the tag.

In a similar fashion hypertext and later HTML provided a presentation environment for viewing electronic information with textual associations called *links*. A lot of confusion has been generated around XML as a replacement for HTML. While this replacement has taken place in HTML 4.0 or XHTML, we need to remember that HTML concentrates on *presenting* data while XML concentrates on *describing* it. That's why XML is ideal for business-to-business communication exchanges. Since XML is text based, it can be used as a transparent data abstraction layer that makes heterogeneous Internet-based computing possible.

XML was also presented as an alternative technology for EDI (Electronic Data Interchange). XML is capable of representing the hierarchical relationships stored inside EDI while at the same time providing a flexible framework for customizing information. EDI implementation proved to be stringent and inflexible as opposed to XML deployment.

The concepts presented by legacy technologies evolve and are reborn in new, fresher technologies. Older technologies need to be used as stepping-stones to validate newer technologies and provide a cohesive baseline for solving even more complex problems. Many of these problems are based on old requirements that need to be supported by new platforms. In similar fashion we can look at new platforms to provide interesting new solutions and to address problems that can't be answered today.

As we look into the future, what are some of the areas in which platforms such as XML can be leveraged and extended to solve new problems? One of them is wireless devices; another is B2B integration and B2B process control flows. These new areas are generating much enthusiasm and will be directly affected by technologies like XML schemas for data validation, XSLT for XML data conversion, XPointer for providing links between XML documents, XPath for addressing or selecting specific portions of an XML document, RDF for providing document content description and BizTalk for providing a specification for process flow management.

One of the technologies in the wireless device area that takes advantage of many of these technologies is Wireless Application Protocol. WAP requires the use of XML schemas: XSLT for output generation to the small screens and JSP (JavaServlet Pages), Xpointer and Xpath to retrieve information stored on regular Web servers and display it on small screens.

These are just some of the technologies that will reshape the future as constantly evolving processing and network devices become part of the mainstream of society.

So where did the parallel between life and technology go? What about Ajit's beliefs that the spirit lives on? The way I see it, technology as well as life has its births and its deaths; however, for those of us who have seen many technologies come and go, we know it's the concept or spirit that continues to make its presence known in everyday life. Similarly, it's my personal belief that the spirit of Ajit's father lives in Ajit and his young son. Therefore, though he will be missed, he will always be with us. Similarly, all of us geeks will continue to see the XML platform grow to become the complete B2B platform that, together with Java, will rule the e-commerce space. ☸

ISRAEL@SYS-CON.COM

AUTHOR BIO

Israel Hilerio is a member of a leading e-commerce firm in Dallas, Texas, focusing on Web-based e-commerce applications and new architectures.

[WRITTEN BY SIET-LENG LAI]

The Internet has become part of our daily lifestyle. With the convergence of communication and computers, and proliferation in the use of the Internet and mobile gadgets, information is readily available in many different forms, such as data, voice, video or even control devices. The control of appliances anywhere, anytime, is no longer the stuff of science fiction. Today the Internet connects not only computing resources and people, but also appliances such as air-conditioning and lighting. The convergence of data and control networks enables corporations and individuals to access all forms of information readily and globally using the Internet connection. With the pervasive use of mobile phones and the introduction of WAP (Wireless Application Protocol) technology, you can easily reach devices such as home appliances via a wireless network and the Internet.

In this article I'll explain how WML (Wireless Markup Language) and Java are used to control or access remote appliances such as lighting, air-conditioning, overhead projectors and card readers using WAP-based gadgets such as mobile phones. This is especially promising in the area of intelligent building. The discussion will be based on the prototype that was built to demonstrate the potential and feasibility of controlling Echelon's LonWorks network of devices. LonWorks platform is the de facto standard, an open universal standard for control networks. It has been adopted by thousands of organizations internationally and has been well received by many different industries.

LonWorks Control Network and Appliances

LonWorks network is a leading, open, networked automation and control solution for the building, industrial, transportation and home markets. There are many applications in these areas, such as intelligent building, asset tracking and facility management, and millions of LonWorks devices have been installed worldwide. In a LonWorks network each control device, which has embedded intelligence, talks to the others using the common LonTalk protocol. This is an open and international standard and the core of LonWorks networks. It is embedded in the form of a neuron chip that implements the protocol and the respective control operations. As shown in Figure 1, the data and control network are seamlessly merged, via a connector gateway such as Coactive's Connector 1000 or Echelon's iLon, and talk to the data network using TCP/IP and to the LonWorks network of appliances using LonTalk protocol. In fact, you can effectively treat appliances such as lamps, air-conditioning and door access devices as part of the IP networks, and you can develop control applications such as home, building, transportation or utility automation to communicate with the intelligent appliances via the gateway.



CONTROL
APPLIANCES
USING
WML
& JAVA
SERVLETS

Access intelligent appliances via mobile gadgets anywhere, anytime

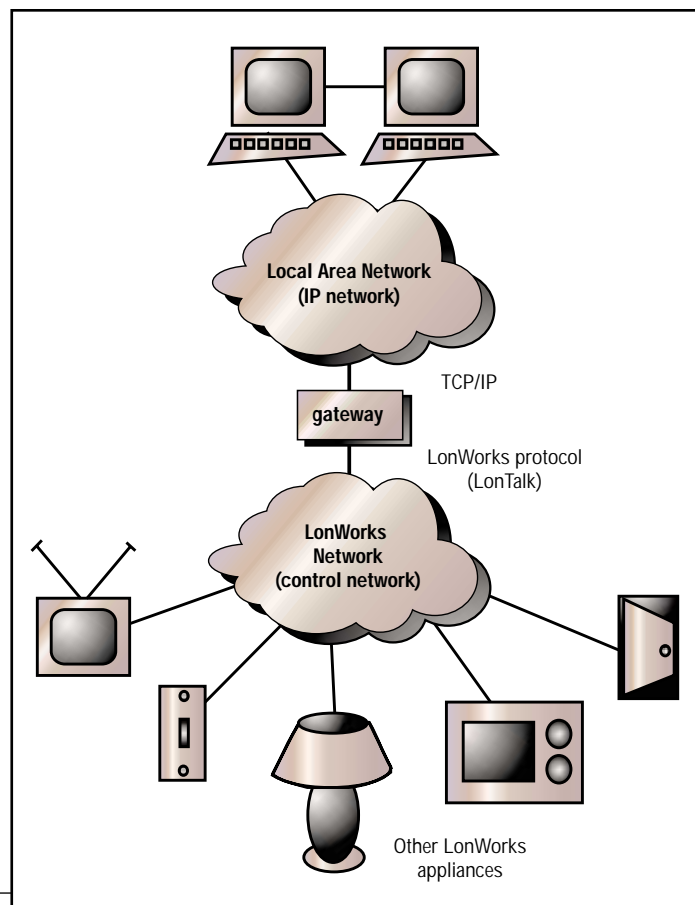


FIGURE 1 Intelligent appliances and the control network

WAP, WML and Appliances

WAP is a protocol for the transmission of data over low-bandwidth wireless networks. With WAP you can bring Internet content and services to the world of mobile and wireless devices, especially mobile phones, which have become indispensable tools and the preferred method of communication as a result of their rapid penetration into all age and professional groups.

WML, part of the application environment of WAP, is a markup language based on XML. It's intended for use in depicting the content and user interface for WAP-based gadgets such as WAP phones. The basic structure of WML is the card, and a number of cards are grouped into decks, which are the root elements of WML documents. A valid WML deck is a valid XML document and thus it comprises XML declaration and DTD (Document Type Definition). The official WML specification is developed and maintained by the WAP Forum, an industry-wide consortium founded by Nokia, Ericsson, Unwired Planet and Motorola. This specification defines the syntax, variables and elements used in a valid WML file whose structure is defined in the WML 1.1 DTD. A WAP-enabled device such as a WAP phone incorporates a microbrowser that can fully interpret and understand the WML specification and can display the results in its display unit.

In this article we'll look at a scenario in which you have a control network of intelligent appliances such as lighting and air-conditioning in your office or at home. You, as a WAP phone user, would like to control/monitor the appliances when you're on the move. For instance, you may want to switch on the air-conditioning before you reach home, monitor the status of the devices in your office or switch off the light at home when you realize that you've forgotten to do so. In other words, you can access the status of the appliances and can carry out different control actions accordingly. This scenario can be further extended via some form of security control – such as a proximity card reader that is part of the control network – that would alert you whenever someone tries to access your home or office. Further authentication or subsequent action can be carried out if necessary.

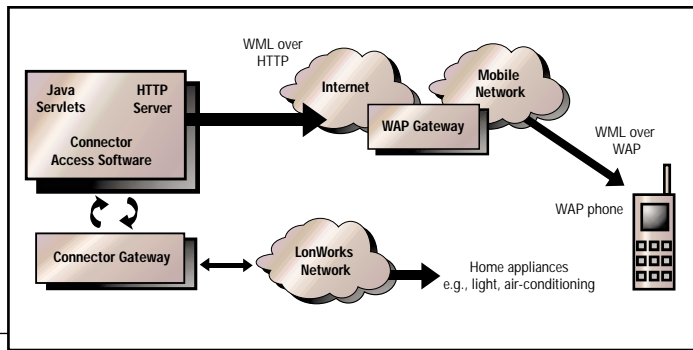


FIGURE 2 Overall architecture of the WAP control application

Anatomy of the WAP Control Application

Figure 2 shows the overall architecture of the WAP-based control application for a network of LonWorks appliances. In developing the prototype, the Java 2 platform was chosen because it's flexible, portable and architecturally neutral. A Java Servlet (JSDK2.1) was developed to provide a server-side solution to communicate with the Coactive connector gateway and the HTTP server (the servlet runner at port 8080 is used in the example). As shown in Figure 2, the servlet communicates with the HTTP server in terms of URL requests and responses in WML codes. On the other side it liaises with the Coactive connector gateway in manipulating the LonWorks appliances.

In developing the prototype, I used the Coactive connector 1000 as a gateway device between the IP and the control network. It provides a set of connector CORBA access software in Java that allows you to have control over the configuration and programming of connector objects in order to talk to the LonWorks network of appliances. You may use any WAP-enabled mobile phone – for example, an Ericsson R320 or a Nokia 7110 – to contact the Java Servlet sitting on your Web server via a WAP gateway as an encoder/decoder. The WAP gateway serves as a middleman between the WAP client – a WAP phone in this case – and the HTTP server that houses the Java Servlet. It encodes and decodes data transferred to and from the WAP phone, resolves the host address specified by the URL and creates an HTTP session to the host. I've been using the operator-controlled WAP gateway during the development of the control application. As shown in Figure 2, two layers of translation, from the LonWorks control network (LonTalk protocol) to the data network (TCP/IP) and from HTTP protocol to WAP protocol – are in place from the WAP user to the appliances.

Figure 3 shows the welcome page on the WAP phone; the corresponding WML code is shown in Listing 1. In this article we'll focus solely on controlling the appliances. Listing 2 shows the essential specification of the servlet class. Some of the detailed methods to interface with the connector gateway are encapsulated to avoid confusion. For instance, individual methods handle functionality like initialization of the ORB for the connector gateway, retrieval of the object reference for the connector objects, and setting and retrieving of the device status. To clarify the subsequent discussion, the WML codes (dynamically generated by the servlet) of the



FIGURE 3 Welcome page for the WAP control

control application to be displayed on the WAP client are extracted and highlighted in Listing 3.

Figure 4 provides a detailed four-phase explanation of the execution of the WAP control application.

Phase 1: With reference to the welcome page shown in Figure 3 and Listing 1, it contains the necessary XML declaration and document type declaration as shown below :

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
...
...
</wml>
```

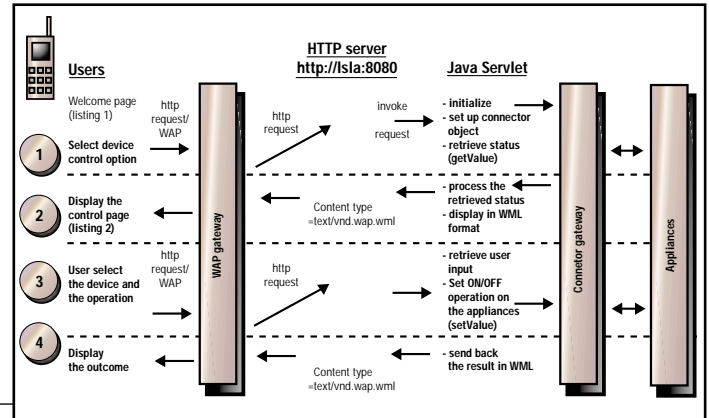


FIGURE 4 Interaction of WAP device, HTTP server, Java Servlet

From the menu shown, you select the device control option. The respective URL request (<http://apricot.ccs.np.edu.sg:8080/servlet/iControl?flag=start>) is then sent to the operator-controlled WAP gateway using WSP (Wireless Session Protocol). After encoding and decoding, the WAP gateway sends the URL request to the Web server using HTTP protocol.

To run the servlet, the servlet runner together with the JSDK2.1 is used as the HTTP gateway at port 8080. On receiving the URL request, the respective Java Servlet – the iControl – is invoked. It will then call the start() method, which includes the initialization process, and set up the connector object to the appliances using software API via the Coactive connector gateway (refer to Listing 2). The preassigned ID of each device is used to retrieve the respective device status using the getValue() method, which contains the necessary invocation to CORBA access software to interact with the connector gateway.

Phase 2: First, the MIME type of the output information must be set.

```
// set the content type to
text/vnd.wap.wml
response.setContentType(
"text/vnd.wap.wml" );
```



FIGURE 5 Control appliances using WAP phone

infoShark
www.infoshark.com

ANNOUNCING...

Wireless Journal

SYS-CON Media, the world's leading publisher of Internet technology magazines for developers, software architects and e-commerce professionals, becomes the first to serve the rapidly growing wireless application development community!

Look for it on your newsstand and worldwide in December!



SYS-CON MEDIA

www.wireless-journal.com

After setting the content type, you can output the declaration and the respective WML statements.

```
// output the declaration and document type definition
    out = response.getWriter();
    out.println("<?xml version=\"1.0\"?>");
    out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML
1.1//EN\"");
    out.println("
http://www.wapforum.org/DTD/wml_1.1.xml\">");
    out.println("<wml>");
    out.println("<card id=\"iControl\" newcontext=\"true\"
title=\"iControl\">");
```

Upon receiving individual device status from phase 1, the information is processed (refer to Listing 3) and presented in WML format :

```
...
<p>
    Light : OFF
    <anchor>ON
    <go href="http://apricot.ccs.np.edu.sg:8080/servlet/iControl"
method="post">
        <postfield name="flag" value="next"/>
        <postfield name="prevPc" value="OFF"/>
        <postfield name="prevAircond" value="OFF"/>
        <postfield name="prevLight" value="OFF"/>
        <postfield name="prevOhp" value="OFF"/>
        <postfield name="light" value="ON"/>
        ...
    </go>
    </anchor>
</p>
...
...
```

The <anchor> element is used to provide the link to toggle the status (ON/OFF) of the selected appliance (e.g., switch on the light). In this prototype the POST method is used and user data is sent using <post-field> tag. The HTTP server will forward the generated WML page to the WAP gateway. The gateway, after encoding and decoding, will send the WML page to the mobile client as shown in Figure 5.

Phase 3: You may select the respective device and the operation (such as ON/OFF) and forward the URL request (POST method is used in this example) to the WAP gateway which will then forward it to the HTTP server by contacting the servlet, iControl. Upon receiving the request, the respective parameters are extracted as follows :

```
pc = request.getParameter("pc");
aircond = request.getParameter("aircond");
ohp = request.getParameter("ohp");
light = request.getParameter("light");

prevPc = request.getParameter("prevPc");
prevAircond = request.getParameter("prevAircond");
prevOhp = request.getParameter("prevOhp");
prevLight = request.getParameter("prevLight");
```

Devices are set – ON/OFF – using the setValue() method via the connector gateway:

```
if (light != null) {
    if (light.equals("ON"))
        setValue(light_value, ON_value);
    else
```

```
setValue(light_value, OFF_value);
```

...

Phase 4: At the end of the operation, the up-to-date WML page is sent back to you via the WAP gateway and is displayed on your WAP phone as shown in Figure 5.

Phases 3 and 4 could be repeated for different operations on different appliances – lighting, air-conditioning, overhead projector.

Conclusion

This article has detailed a prototype WAP-based control application device using WML and Java servlets. Though still in its infancy, and with room for improvement, I have demonstrated the potential and feasibility of using gadgets such as mobile phones to control a network of remote appliances. WAP technology opens up many interesting opportunities for different industry sectors in terms of innovative and useful application domains. One possible application area is intelligent building that integrates telecommunications, information networks and building/home automation to provide convenient and easy access to control information. This makes it possible to control/access intelligent appliances via mobile gadgets anywhere, anytime. ☉

References

1. *Wireless Application Protocol Architecture Specification*, WAP Forum: www.wapforum.org/
2. *Wireless Markup Language Specification*, WAP Forum: www.wapforum.org/
3. *Echelon LonWorks Networks*: www.lonworks.echelon.com/
4. *Coactive Connector 1000*, Coactive Networks Inc.: www.coactive.com/
5. *Nokia*: www.nokia.com/
6. *Ericsson*: www.ericsson.com/
7. *Unwired Planet*: www.phone.com/
8. *Motorola*: www.motorola.com/
9. *Java 2 Platform*, JavaSoft: www.javasoft.com/

AUTHOR BIO

Siet-Leng Lai has published and presented at several international conferences, and is now conducting research into XML, WML and WAP. Since 1990 she has been a lecturer at the Center for Computer Studies, Ngee Ann Polytechnic, Singapore, on subjects ranging from operating systems, system programming, networking, Internet computing, and parallel and distributed computer systems to intelligent building. Lai has also conducted short courses for the public on the above-mentioned topics. She holds degrees from the National University of Singapore.

LSL@NP.EDU.SG

LISTING 1

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
  <card id="Welcome" title="Intelligent Services">

    <p>
      <b>What service ?<br/></b>
      <a href="http://apricot.ccs.np.edu.sg:8080/servlet/iControl?flag=start">Control Devices
    </a><br/>
      <a href="http://apricot.ccs.np.edu.sg:8080/servlet/iStatus">Device status </a><br/>
      <a href="http://apricot.ccs.np.edu.sg:8080/servlet/Booking">Facility Booking</a><br/>
    </p>
  </card>
</wml>
```

LISTING 2

```
import java.io.*;
import java.text.*;
import java.util.*;
```

Why use a spy, when you can be more productive with your Mate?



**XMLMATE™ from in Insight
is the most effective
XML converter and XML
and DTD editor available.**

Find out today what XMLMATE
can deliver for you.

For your **FREE** 30 day trial,

Visit

www.xmlboutique.com

XMLMATE brought to you by:




```

import javax.servlet.*;
import javax.servlet.http.*;

import java.lang.*;

// to be used for Coactive control access software
import org.omg.CORBA.TCKind;

public class iControl extends HttpServlet {

    PrintWriter out;
    String pc, aircond, ohp, light;

    /* values defined for LonWorks devices */
    static int pc_value=44;
    static int aircond_value=45;
    static int light_value=51;
    static int ohp_value=52;

    static String ON_value = "1";
    static String OFF_value = "0";
    . . .
    static String ior = null;
    static String ip = null;
    org.omg.CORBA.ORB orb = null;
    . . .

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {

        String flag;
        String prevCardreader, prevPc,
            prevAircond, prevOhp, prevLight;

        flag = request.getParameter("flag");

        // set the content type for WML
        response.setContentType("text/vnd.wap.wml");

        // output WML header
        out = response.getWriter();
        out.println("<?xml version='1.0'?>");
        out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
        out.println("<http://www.wapforum.org/DTD/wml_1.1.xml>");
        out.println("<wml>");
        out.println("<card id='iControl'");
        out.println("newcontext='true' title='iControl'>");

        // if this is the first time, retrieve the status of the
        appliances i.e. PC, Air-Cond, Light and OHP
        if (flag.equals("start")) {
            start();
        } else {

            pc = request.getParameter("pc");
            aircond = request.getParameter("aircond");
            ohp = request.getParameter("ohp");
            light = request.getParameter("light");

            prevPc = request.getParameter("prevPc");
            prevAircond = request.getParameter("prevAircond");
            prevOhp = request.getParameter("prevOhp");
            prevLight = request.getParameter("prevLight");

            // ON/OFF operation for the respective appliances

            if (pc != null) {

                if (pc.equals("ON"))
                    setValue(pc_value, ON_value);
                else
                    setValue(pc_value, OFF_value);

            } else
                pc = prevPc;

            if (aircond != null) {
                if (aircond.equals("ON"))
                    setValue(aircond_value, ON_value);
                else
                    setValue(aircond_value, OFF_value);
            } else
                aircond = prevAircond;

            if (light != null) {

```

```

                if (light.equals("ON"))
                    setValue(light_value, ON_value);
                else
                    setValue(light_value, OFF_value);
            } else
                light = prevLight;

            if (ohp != null) {

                if (ohp.equals("ON"))
                    setValue(ohp_value, ON_value);
                else
                    setValue(ohp_value, OFF_value);
            } else
                ohp = prevOhp;
        } // next

        StringBuffer ref_str = new StringBuffer();

        ref_str.append("<go href='http://apricot.ccs.np.edu.sg:8080/servlet/iControl' method='post'> \n");
        ref_str.append("<postfield name='flag'");
        value="\next\" /> \n");
        ref_str.append("<postfield name='prevPc'");
        value="\"+pc+"\" /> \n");
        ref_str.append("<postfield name='prevAircond'");
        value="\"+aircond+"\" /> \n");
        ref_str.append("<postfield name='prevLight'");
        value="\"+light+"\" /> \n");
        ref_str.append("<postfield name='prevOhp'");
        value="\"+ohp+"\" /> \n");

        out.println("<p>");
        out.println("PC : "+pc);

        if (pc.equals("ON")) {
            out.println(" <anchor>OFF");
            out.print(ref_str);
            out.println("<postfield name='pc' value='OFF' />");
            out.println("</go>");
            out.println("</anchor>");
        } else {
            out.println(" <anchor>ON");
            out.print(ref_str);
            out.println("<postfield name='pc' value='ON' />");
            out.println("</go>");
            out.println("</anchor>");
        }

        out.println("</p>");
        out.println("<p>AirCond.: "+aircond);

        temp_str = new StringBuffer(ref_str.toString());

        if (aircond.equals("ON")) {
            out.println(" <anchor>OFF");
            out.print(ref_str);
            out.println("<postfield name='aircond' value='OFF' />");
            out.println("</go>");
            out.println("</anchor>");
        } else {
            out.println(" <anchor>ON");
            out.print(ref_str);
            out.println("<postfield name='aircond' value='ON' />");
            out.println("</go>");
            out.println("</anchor>");
        }

        out.println("</p>");

        out.println("<p>Light : "+light);

        if (light.equals("ON")) {
            out.println(" <anchor>OFF");
            out.print(ref_str);
            out.println("<postfield name='light'");
            value="\next\" /> \n");
            out.println("</go>");
            out.println("</anchor>");
        } else {
            out.println(" <anchor>ON");
            out.print(ref_str);
            out.println("<postfield name='light'");
            value="\next\" /> \n");
            out.println("</go>");
            out.println("</anchor>");
        }

        out.println("</p>");
    }
}

```

```

        out.println("<p>OHP : "+ohp);

        if (ohp.equals("ON")) {
            out.println(" <anchor>OFF");
            out.print(ref_str);
            out.println("<postfield name=\"ohp\" value=\"OFF\" />");
            out.println("</go>");
            out.println("</anchor>");
        } else {
            out.println(" <anchor>ON");
            out.print(ref_str);
            out.println("<postfield name=\"ohp\" value=\"ON\" />");
            out.println("</go>");
            out.println("</anchor>");
        }

        out.println("</p>");
        out.println("</card>");
        out.println("</wml>");
    }

    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
        doGet(request, response);
    }

    private void start()
    {
        // IP address of the Coactive connector gateway
        ip = new String("153.20.248.29");
        ior = genior();

        // initialize the orb for reference to the connector
        // gateway
        if (initOrb() == 1)
        {
            System.err.println("init(): initOrb() failed ");
        }
        else
        {
            System.err.println("init(): initOrb() succeeded ");
        }

        // get references to objects available on the Connector
        if (getConnection() == 1)
        {
            System.err.println("init(): getConnection() failed ");
        }
        else
        {
            System.err.println("init(): getConnection() succeeded ");
        }
        getDefs();

        // retrieve current device status for PC, Aircond, light and ohp
        String pc_string = getValue(pc_value);
        String aircond_string = getValue(aircond_value);
        String light_string = getValue(light_value);
        String ohp_string = getValue(ohp_value);

        if (pc_string.equals(ON_value)) {
            pc = "ON";
        } else {
            pc = "OFF";
        }

        if (aircond_string.equals(ON_value)) {
            aircond = "ON";
        } else {
            aircond = "OFF";
        }

        if (light_string.equals(OFF_value)) {
            light = "ON";
        } else {
            light = "OFF";
        }

        if (ohp_string.equals(ON_value)) {
            ohp = "ON";
        } else {
            ohp = "OFF";
        }
    } // end start

    void setValue(int dpId, String value_to_set)
    {
        ...
    }

```

```

// to control the device according to the operation
// specified e.g. ON/OFF
...
}

String getValue(int dpid)
{
    ...
    // to retrieve the status of the device ... return as a string
    ...
}

...

} // servlet class definition

```

LISTING 3

```

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<!-- Source Generated by WML Deck Decoder -->
<wml>
    <card id="iControl" newcontext="true" title="iControl">
        <p>
            PC : OFF
            <anchor>ON
            <go
href="http://apricot.ccs.np.edu.sg:8080/servlet/iControl"
method="post">
                <postfield name="flag" value="next"/>
                <postfield name="prevPc" value="OFF"/>
                <postfield name="prevAircond" value="OFF"/>
                <postfield name="prevLight" value="OFF"/>
                <postfield name="prevOhp" value="OFF"/>
                <postfield name="pc" value="ON"/>
            </go>
            </anchor>
        </p>
        <p>
            AirCond.: OFF
            <anchor>ON
            <go
href="http://apricot.ccs.np.edu.sg:8080/servlet/iControl"
method="post">
                <postfield name="flag" value="next"/>
                <postfield name="prevPc" value="OFF"/>
                <postfield name="prevAircond" value="OFF"/>
                <postfield name="prevLight" value="OFF"/>
                <postfield name="prevOhp" value="OFF"/>
                <postfield name="aircond" value="ON"/>
            </go>
            </anchor>
        </p>
        <p>
            Light : OFF
            <anchor>ON
            <go
href="http://apricot.ccs.np.edu.sg:8080/servlet/iControl"
method="post">
                <postfield name="flag" value="next"/>
                <postfield name="prevPc" value="OFF"/>
                <postfield name="prevAircond" value="OFF"/>
                <postfield name="prevLight" value="OFF"/>
                <postfield name="prevOhp" value="OFF"/>
                <postfield name="light" value="ON"/>
            </go>
            </anchor>
        </p>
        <p>
            OHP : OFF
            <anchor>ON
            <go
href="http://apricot.ccs.np.edu.sg:8080/servlet/iControl"
method="post">
                <postfield name="flag" value="next"/>
                <postfield name="prevPc" value="OFF"/>
                <postfield name="prevAircond" value="OFF"/>
                <postfield name="prevLight" value="OFF"/>
                <postfield name="prevOhp" value="OFF"/>
                <postfield name="ohp" value="ON"/>
            </go>
            </anchor>
        </p>
    </card>
</wml>

```





Diving into the SOAP specification

Part 2

SOAP

The last edition of the **XML in Transit** column (*XML-J*, Vol. 1, issue 4) introduced the Simple Object Access Protocol (SOAP). Instead of dwelling on technical issues, it focused on the driving forces behind the technology.

To put SOAP into context we looked at its history, parsed the buzzword-compliant phrase *ubiquitous XML distributed computing* infrastructure and scoped the SOAP specification within the broader set of standards likely to emerge in this area. One possible layering of SOAP and related specifications is shown in Figure 1. The areas already covered by SOAP are grayed out. As far as protocol support is concerned, the latest SOAP specification (SOAP 1.1) provides bindings only for HTTP.

In this column we'll go deep into the SOAP specification. Let's start our dive into the technical details with a look at SOAP's extensibility framework.

Extensibility Framework

XML has become the lingua franca for structured data exchange on the Internet. In particular, with the wider adop-

tion of XML Namespaces and XML Schemas we now have the ability to define and extend XML protocols in a distributed environment. In other words, it's now possible for a single organization or standards body to create a specification and publish it for broad use while also guaranteeing that it can be safely extended at a later date by independent third parties. *Safely* in this context means that (1) optional extensions shouldn't break existing applications, (2) mandatory extensions will work only with applications that understand them and (3) multiple extensions can coexist peacefully.

Don't get me wrong – I said it was possible, not trivial, to do this. It's not the case that schemas and namespaces automatically enable this type of interoperability. However, they make it possible for mere mortals to develop speci-

cations and extensions with these properties, and this is important enough. Think about the last time you tried to extend some flat data model or even tried to add a couple of elements to your XML DTD. Could your existing software safely skip over the optional extensions? Or did you have to change your code? Be honest. No one's looking.

Simplicity and extensibility are major design goals for SOAP. They're achieved through modularity and layering. SOAP is designed to maintain different aspects of the specification as completely independent or "orthogonal."

Here's how several SOAP facets are treated:

- **Communication protocols:** Generally speaking, the choice of communication protocol has no impact on the structure and content of SOAP messages. SOAP is protocol-independent. However, the SOAP specification details how SOAP messages can be sent over HTTP because it's likely that HTTP will be the most common delivery mechanism for SOAP messages. We can also be sure that SMTP will be a common mechanism for simple message passing and that pure sockets are going to be used frequently inside firewalls for performance.
- **Communication patterns:** The choice of communication patterns (request-response, point-to-point, publish-subscribe) may change the structure of a SOAP message, but the schemas identified by the SOAP-specific namespaces don't need to be changed. SOAP messages are a mixture of elements from SOAP-specific namespaces and any other namespaces. The choice of communication pattern may impact the schema design within these other namespaces. In other

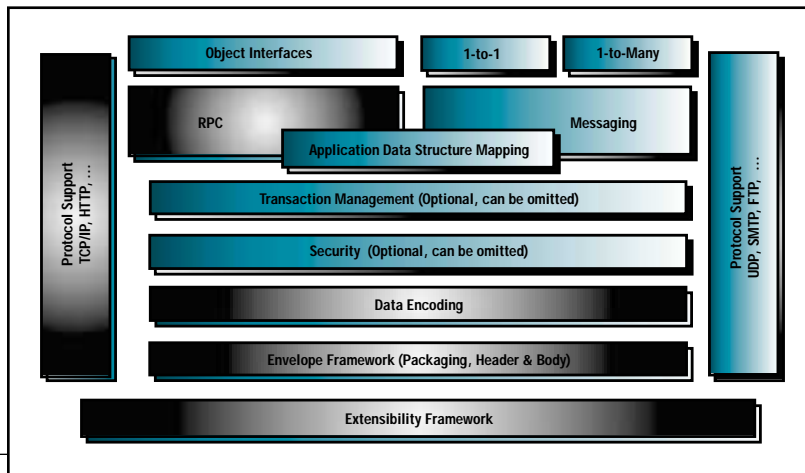


FIGURE 1 One possible layering of SOAP and related specifications

Cape Clear

www.capeclear.com

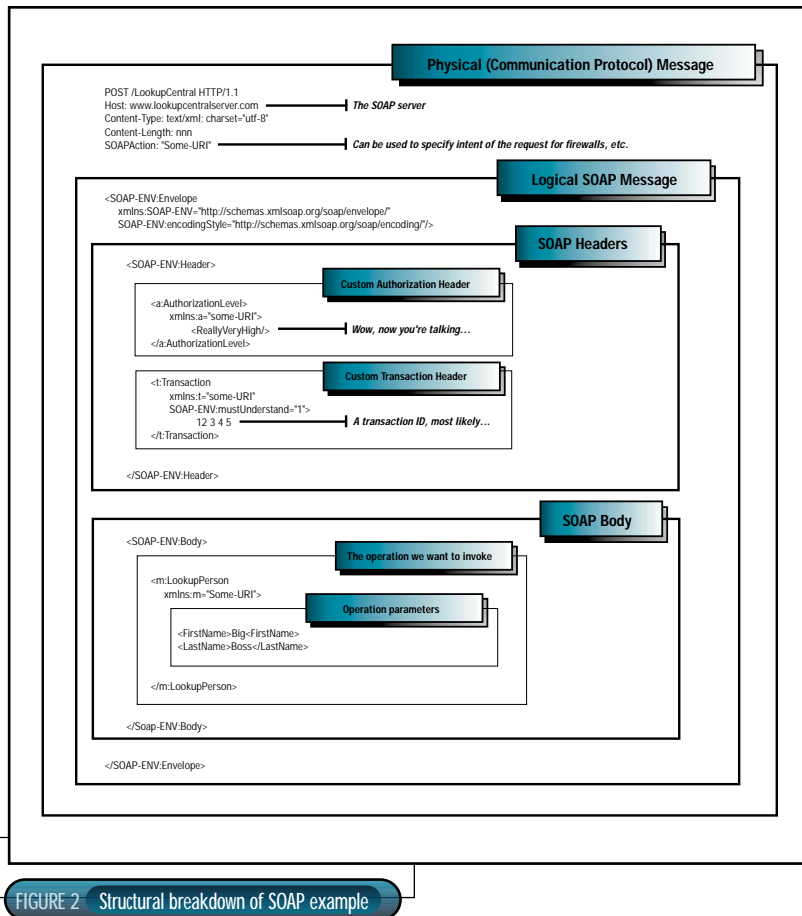


FIGURE 2 Structural breakdown of SOAP example

words, "application-level" schemas might change, but the core SOAP schemas stay intact. SOAP defines one possible way to do RPC, but others can be provided.

- SOAP Envelope:** According to the SOAP 1.1 specification, "the SOAP Envelope construct defines an overall framework for expressing what's in a message, who should deal with it, and whether it's optional or mandatory." All this is done with just a few XML elements, as we'll see later in this column. The schema is kept in a separate namespace (<http://schemas.xmlsoap.org/soap/envelope> aliased as SOAP-ENV in the specification) so it doesn't affect other parts of SOAP. The envelope framework allows any "application-level" XML to be packaged as headers or the body of a SOAP message. It's the only SOAP schema that can't be replaced.
- Data encoding:** SOAP defines one particular set of data encoding rules. Again, they're kept in their own namespace (<http://schemas.xmlsoap.org/soap/encoding> aliased as SOAP-ENC in the specification). The SOAP implementation by the Apache XML Project already has an alternative encoding based on XML. Any other data encoding rules can be provided – for example, Web

Don't Miss JDJ's LINUX Focus Issue!



COMING IN DECEMBER

Java Developer's Journal

Announces Special LINUX Focus Issue

Insertion Order Due: **OCTOBER 19, 2000**

Artwork at Our Press: **OCTOBER 26, 2000**

For Advertising Information Contact:

Carmen Gonzalez

Vice President, Advertising Sales
Java Developer's Journal

Call **(201) 802-3021**

or email carmen@sys-con.com



www.JavaDeveloper'sJournal.com

Distributed Data Exchange (WDDX). Any portion of a SOAP message can have its own encoding rules, which can be intermixed freely. The SOAP-ENV: encodingStyle attribute binds encoding rules to an element and its content. The value of the attribute is the namespace for the encoding.

Now that you understand the “pretty much anything goes” policy of SOAP, let’s look at a detailed example. We’ll pick it apart to learn more about the envelope framework.

SOAP Message Structure

Listing 1 is an example request to look up the name “Big Boss” using a SOAP, server managed by Lookup Central. In this case we’re using SOAP to implement an RPC over HTTP mechanism.

This is a fairly complex SOAP example. To get a better handle on it, I’ve broken it down structurally in Figure 2 and added some comments.

The first thing that’s apparent is the layering of logical on top of physical protocols. The XML of the logical SOAP message is contained within a physical HTTP request. (Yes, I do agree that the notion of logical versus physical protocols gets quickly confused when one thinks about a thick protocol stack, but

please bear with me here.) There are a couple of important things to note about this:

- The location of the SOAP server (www.lookupcentral.com/Lookup-Central) is identified within the HTTP request; it’s not part of the SOAP message. By being addressing-agnostic SOAP can be protocol-independent.
- The optional SOAPAction header of the HTTP request identifies the intent of the SOAP request. Its value can be any URI. For example, the value can be used by firewalls to appropriately filter SOAP messages.

When we look at the beginning of the SOAP message, we see the SOAP Envelope element. It’s the mandatory top element of the XML document representing the message, and because it’s uniquely identified by its namespace, it allows processing tools to immediately determine whether a given XML document is a SOAP message.

This certainly is convenient, but what do we trade off for this capability? The biggest thing we have to sacrifice is the ability to send arbitrary XML documents and perform simple schema validation on them. True, we can embed arbitrary XML inside the SOAP Body element, but naïve validation will fail

when it encounters the Envelope element at the top of the document instead of the top document element of our schema. The lesson is that for seamless validation of arbitrary XML inside SOAP messages, we need smarter tools.

A final note about the Envelope element: it doesn’t expose any protocol versioning information (such as a major and minor version number). Instead, the notion of versioning is tied to the namespace URI. As the protocol evolves, the URI is likely to change.

Going further with the example message, under the Envelope element we see a Header element. Headers, which are optional in SOAP, are the primary mechanism by which additional facets can be added to SOAP-based protocols. They provide an elegant yet simple mechanism to extend SOAP messages in a decentralized manner. Typical areas where headers get involved are authentication and authorization, transaction management, payment processing, tracing and auditing. Another way to think about this is that you’d use headers to pass any information orthogonal to the specific information needed to execute a request.

For example, a stock quote service only needs a stock symbol to generate a

TheShortList

www.theshortlist.com

quote. In real-world scenarios, however, a stock quote request is likely to contain much more information (e.g., the identity of the person making the request, account/payment information). This additional information is usually handled by infrastructure services (login and security, transaction coordination, billing) outside the main stock quote service. Encoding this information as part of the body of a SOAP message will only complicate matters. That's why it will get passed in as headers.

All immediate children of the Header element are header entries. In Figure 2 we have two. The first one has something to do with the authorization level of the entity making the request. The second passes what looks like a transaction identifier to the SOAP server. In general, header entries can have arbitrary XML in them. Note how the unrelated header entries use separate namespaces.

Also, notice the SOAP mustUnderstand attribute with value "1" that decorates the Transaction element. This attribute indicates that the recipient of the SOAP message must process the Transaction header entry. If a recipient doesn't know how to process a header tagged with mustUnderstand="1" it must abort processing with a well-defined error. This rule allows for robust evolution of SOAP-based protocols. It ensures that a recipient that might be unaware of certain important protocol extensions doesn't ignore them.

Because the AuthorizationLevel header entry isn't tagged with mustUnderstand="1", it can be ignored during processing. Presumably this will be okay because a server that doesn't know how to process authorization levels will operate under a safely low authorization level, one that doesn't allow intruders to gain access to confidential information (in this case, perhaps the salary of the Big Boss).

I'd like to finish the discussion of headers with a reminder that the two headers in the example are fully fictitious. There has been no broad consensus in the SOAP community about how to handle authentication, authorization or transactions. We need these to enable enterprise-quality SOAP-based systems.

The SOAP Body element immediately surrounds the information that's intended for the final recipient of the SOAP message. Truth be told, you can pass the same information using a header entry tagged with mustUnderstand="1", but using a mandatory Body element just makes things easier to understand and process.

All immediate children of the Body

element are body entries. In the example, we want to perform an RPC request to the LookupCentral server and look up the name "Big Boss" in it. Although arbitrary XML can be passed via a body entry, by convention, in many cases of RPC over SOAP the name of the first body entry element identifies the method you're interested in accessing. In this case, the LookupPerson element identifies a method with the same name. Note the use of yet another namespace to keep method-related information separate from everything else.

The content of LookupPerson is the set of parameters we need to pass to the method to do its job. In the example we're passing two strings – the first and last names of the person we're looking up. These are simple values, but keep in mind that SOAP can encode pretty much any data structure.

That's most of what there is to a SOAP message structure. Let's look at a possible response message.

The SOAP Response

As you can see in Listing 2, the structure of the SOAP response message is identical to that of the SOAP request message.

The Transaction header entry is returned to indicate that the response is part of the same transaction as the request. The AuthorizationLevel header entry is absent because it makes sense only during the request. The Body element contains a LookupPersonResponse element. Adding "Response" to the request element name is a convention suggested (but not required) by the SOAP specification. The response data is encoded via the SOAP encoding style used by the request. We can see that Big Bad Boss is CEO of UberSoft Corporation and that she has a nice round salary of \$10,000,000. That's all there is to it.

SOAP Message Exchanges

On the surface SOAP message processing seems really simple. A SOAP server will get a message, check that it knows how to process all elements tagged with mustUnderstand="1" and then invoke whatever back-end functionality the message requests. Actually, there's a lot more going on. The fact that SOAP is independent of the choice of protocol and communication pattern is great from the perspective of simplicity and flexibility; however, it means that SOAP clients and servers must maintain separate conventions on how to perform message exchanges and that takes some effort. Let's look at the example in a little more detail.

The SOAP message arrives as an

HTTP request. The SOAP server that's the recipient of the message is identified at the HTTP protocol level (www.lookupcentral.com/LookupCentral). The SOAP client needs to know (1) that there's a SOAP server at that URL and (2) how to send a message to it. This means sending an HTTP request to the URL with ContentType text/xml, with the message as the body of the request.

The SOAP server needs to be listening for messages at that URL and, although SOAP messages can be thought of as one-way transmissions from a sender to a receiver, in this particular example both the SOAP client and server must know that the response message will be part of the HTTP response. Successful message processing should return HTTP status 200 OK. The bottom line is that there needs to be a lot of shared knowledge about the protocol over which the client and the server must communicate.

The same is true at the logical protocol level. Both client and server need to know that they are engaging in an RPC communication pattern. Both must use the convention that the name of the first body entry element specifies the operation that needs to be performed. Both must understand the encoding style used to represent information.

Other semantics need to be agreed on. For example, the client needs to encode transaction information into a header entry. The server needs to be able to process this transaction information (the Transaction element is tagged with mustUnderstand="1"). And the server needs to know that the response message should include the same transaction ID as the request message.

As you can see, for meaningful message exchanges to occur over SOAP, clients and servers need to agree on a lot of things that are outside the scope of the SOAP specification. That's why it's so important for the industry to quickly address the broad set of problems in the area of XML-based distributed computing as shown in Figure 1. I'll spend more time looking at these issues in future installments of this series.

I'll complete the SOAP trilogy with SOAP Part 3, which will address aspects of the latest specification we haven't covered so far: data encoding, error handling and the really cool concept of intermediaries. Once again, SOAP will prove simple, elegant and flexible, and far short of providing the full solution. Stay tuned for details. ☛

AUTHOR BIO

Simeon Simeonov is chief architect at Allaire Corporation.



LISTING 1

```
A SOAP message example
POST /LookupCentral HTTP/1.1
Host: www.lookupcentralserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/">
  <SOAP-ENV:Header>
    <a:AuthorizationLevel
      xmlns:a="some-URI">
      <ReallyVeryHigh/>
    </a:AuthorizationLevel>
    <t:Transaction
      xmlns:t="some-URI"
      SOAP-ENV:mustUnderstand="1">
      12345
    </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:LookupPerson
      xmlns:m="Some-URI">
      <FirstName>Big</FirstName>
      <LastName>Boss</LastName>
    </m:LookupPerson>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

LISTING 2

```
A SOAP message response example
HTTP/1.1 200 OK
```

```
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
```

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encod-
ing/">
  <SOAP-ENV:Header>
    <t:Transaction
      xmlns:t="some-URI"
      SOAP-ENV:mustUnderstand="1">
      12345
    </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:LookupPersonResponse
      xmlns:m="Some-URI">
      <FirstName>Big</FirstName>
      <MiddleName>Bad</MiddleName>
      <LastName>Boss</LastName>
      <Company>UberSoft Corporation</Company>
      <Position>CEO</Position>
      <Salary>
        <Currency>USD</Currency>
        <Amount>10000000</Amount>
      </Salary>
    </m:LookupPersonResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



iXiasoft

www.ixiasoft.com

[WRITTEN BY MAHESH CHULET]

The Wireless Application Protocol (WAP) specification defines Wireless Markup Language (WML) and Wireless Markup Language Script (subsets of the Wireless Application Environment [WAE] specification) as the languages for displaying and adding procedural logic to the content on a WAP-enabled wireless device.

In WAP terminology WML documents are referred to as *decks*. A deck contains procedures for user interaction in the form of cards. This article familiarizes you with WML, the structure of a WML document and the various components associated with it.

WAP is a specification developed by the WAP forum. It's a set of communication protocols that standardize the mechanism used by wireless devices to access the Internet. The WAP forum is an industry association comprising of more than 200 members. These members constitute 95% of the global handset and carrier market. Big names in this market include Alcatel, AOL, Motorola, Microsoft and AT&T.

The WAP forum also defines an industry-wide specification for developing applications and services for wireless devices, the WAE (see Figure 1).

The WAE consists of the following entities:

- WML microbrowser
- WMLScript virtual machine
- WMLScript standard library
- Wireless telephony application interface
- WAP content types

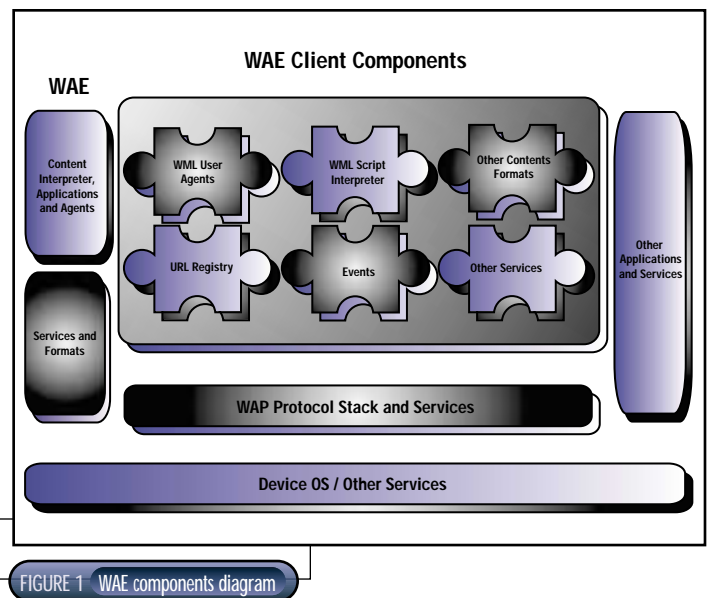
The WML and WMLScript entities of the WAE application layer protocol specification deal with content display and client-side scripting.

WAP Content

The HTML's rich display capability is unsuitable for wireless devices due to low computing power, a limited memory, a small display, an input mechanism limited by a numeric keypad, limited bandwidth wireless channels and a device used mostly with one hand. The WAP specification implementation tries to make optimum use of small display units, to allow navigation to be performed with one hand and to provide the maximum data in the smallest packet size. Navigation mechanisms in WAP include HTML-type hyperlinks and intercard navigation elements (tags). These mechanisms also keep track of history navigation elements, which are similar to navigation mechanisms provided over the Web using HTML. They also define the variables, elements and syntax to be used in a valid WML file, thus standardizing the language for developing WAP content.

WML and WMLScript are analogous to HTML and JavaScript. The former is a tag-based document language used for displaying content on wireless devices; the latter is a scripting language that adds client-side procedural logic to the WML decks. WML inherits properties and behavior from HTML and HDML (Handheld Device Markup Language). It's quite a strict language in comparison to HTML as it's based on XML. Though WML isn't really a subset of HTML, they do share some common tags. Compared to HTML, WML is lightweight with fewer tags, and

Develop applications and services for wireless devices



instead of creating pages you create decks that consist of one or more cards. Decks are WML documents that contain cards – procedures for particular user interactions.

WMLScript is based on the ECMA script standard. It's a weakly typed language; the variables' data types are determined when the variable is assigned a value. The data types for supported variables are boolean, integer and floating point variables. WMLScript supports an if...then...else set of flow control structures. The naming rules for WMLScript variables are similar to the ones for WML, namely:

- Alphanumeric characters can be used.
- Underscore can be used within names.
- All the names are case sensitive.
- Names shouldn't begin with a digit.

WML is encoded, while WMLScript is compiled into binary form before it's sent to the client. WMLScript needs to be compiled into the WMLScript bytecode before it can be run on a WAP client (e.g., a WAP phone). Unlike HTML and JavaScript, WML contains references to WMLScript URLs only. The WAP client must contain a WMLScript virtual machine to run the compiled script. Thus it provides a standard means for applying procedural logic to WML decks. WAE specifies a set of standard library functions. These functions should be available on platforms seeking compliance with the WAP specification.

Communication

A user wishing to access a particular service from a WAP server submits a request to the server using a WML user agent (e.g., microbrowser). The user agent sends a URL request to the WAP gateway by using the WAP protocols. The request for service is done in a manner similar to the HTTP GET request. Once the WAP server receives a request, it makes a request to a Web site or to any other back-end system for the needed resource. After the result is obtained from the external resource, the content is processed by a gateway. The gateway converts this content into a format that's best suited for limited bandwidth transmission and limited-device processing capability. Eventually it transmits this information in an optimized format (see Figure 2).

The gateway and the request processing server can be a single system. The gateway doesn't always need to be present in a WAP network. WML encoding and WMLScript compilers can be built into the WAP server. The user agent receives the WAP response, parses the WML content and displays the first card of the WML deck to the user. If the user agent comes across a reference to WMLScript while interpreting the WML content, it sends out a request to obtain these scripts. On receiving the WMLScript from the server, the WMLScript VM in the user agent interprets and executes it.

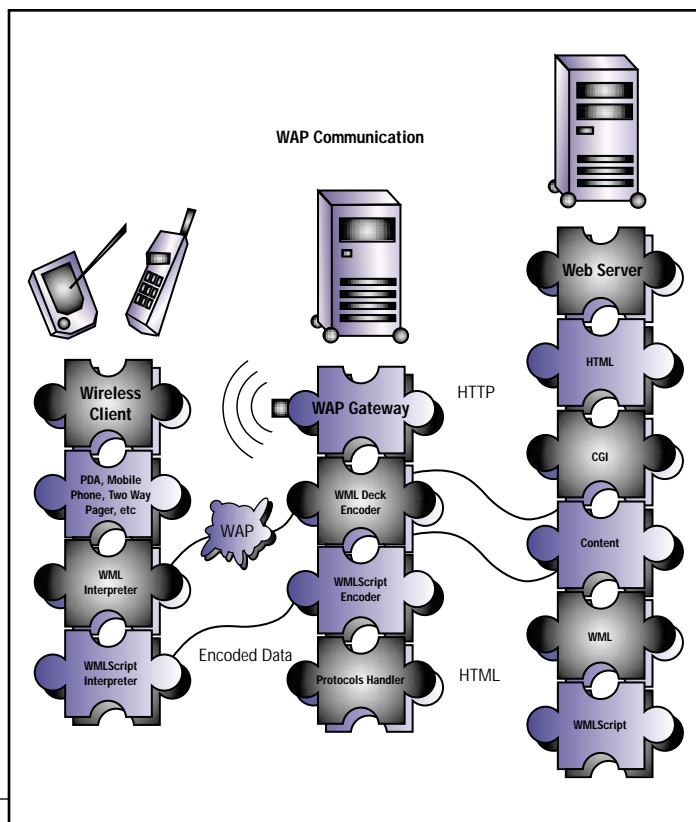


FIGURE 2 WAP communication

WML Details

DOCUMENT STRUCTURE

WML is a tag-based, document-formatting language that implements a subset of HDML version 2.0. It inherits most of its syntactic constructs from XML and HTML. Like XML, it's case sensitive, which implies that all the tags, attributes and contents are case sensitive too. WML tags enable content providers to specify text and images that can be displayed by a wireless device. The content is surrounded by tags, and each tag is enclosed in a pair of angle brackets. The start tag can have attributes associated with it (`<card id="one">`) that provide extra information, similar to tag operations in HTML and XML. In WML the new tag format is the empty element, as compared to HTML. These tags have the open and close tags merged into a single tag (`<tag/>`). Comments in WML follow the same syntax as HTML and XML, and the user agent doesn't display them to the user. The syntax for comment is `<!--` an ancient comment tag `-->`.

WML pages, documents or decks can contain single or multiple cards. The start of a deck is identified by the opening `<wml>` tag and the end by the closing `</wml>` tag. Numerous cards can be placed between them. WML cards are similar to the destination anchors in HTML documents, which may be specified by the `<A>` element (identifying it with the name attribute). The start of a card is identified by the opening `<card>` tag and the end by the closing `</card>` tag. Cards specify one or more units of user interaction – for example, a screen of text, a text entry or a field choice menu. Cards can be labeled and grouped together in a deck. The procedural logic coded in WMLScript, or the reference tags in WML embedded within cards, may invoke services on the originating server as needed by a particular deck's interaction. Listing 1 provides a basic WML deck example.

HTML and XML developers must feel comfortable with the syntax of the preceding WML deck example. Every WML deck begins with a header statement introducing the document (this should be incorporated into every WML page). The next line specifies that a valid WML deck is a valid XML document and therefore contains both XML and document-type declarations. The `<wml>` and `</wml>` tags bind the WML

deck content. The `<card>` and `</card>` tags bind the content of a WML card. Cards help divide the deck into modules for user interaction; usually only a single card is displayed. After the user finishes interacting with a card, he or she can move to the next card in the deck using the navigational links provided.

Another high-level entity in the WML deck is the template. A template defines deck-level characteristics that apply to all cards in a deck. This provides a great mechanism for adding default behavior to your WML cards. Individual cards that need to provide different characteristics can override the ones defined in the template by declaring characteristics using the same name.

ENTITIES

WML contains tokens that are used by the WML interpreter to introspect content attributes. Using these tokens in regular text will confuse the WML interpreter and provide incorrect output. These characters must be represented by entities in order to be used in regular content. When entities are interpreted, they substitute characters that represent tokens. Use the `<` entity because the less than sign, "`<`", is used to begin tags, such as `<wml>`. If you used "`<`" in text programs that read WML documents, the programs would be unable to determine if it's a tag or regular content. All entities begin with an ampersand and end with a semicolon. WML text can contain numeric or named character entities. See Figure 3 for an example of WML text.

TEXT FORMATTING

WML supports HTML elements for text formatting. The various tags provided are:

- **Text emphasis elements:** For example, small, big, u, b, i, strong and em
- **br element:** For new line
- **p element:** For paragraph-type formatting
- **Table element:** For creating tables, including the tr and td elements

EVENTS AND TASKS

WML provides quite a few elements that allow you to handle events and navigation by processing the user agent events. This is similar to the callback mechanism that's used when programming with high-level languages. In WML you can associate a task with an event, so when the specified event occurs, the attached task will be executed. Event binding with a task can be done with the help of several elements provided in WML, such as `do`, `onevent` and `ontimer`. Specify the tasks to be executed on a particular event by using the following WML elements: `go`, `prev`, `refresh` and `noop`.

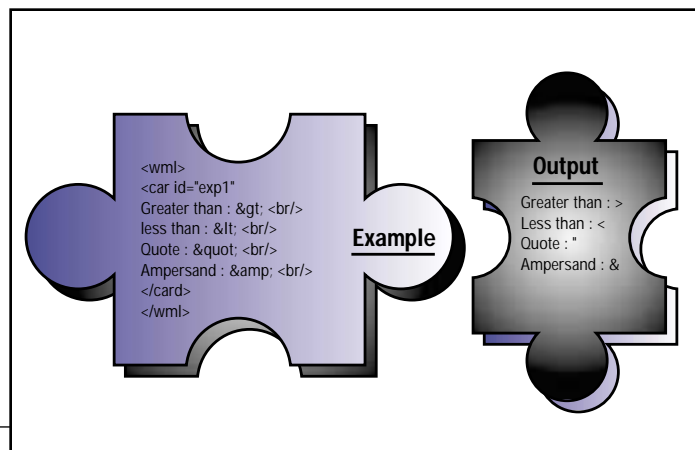


FIGURE 3 Output display obtained for the code using WinWAP WML browser

VARIABLES

WML decks and cards accept parameters that can be set using variables. The variable syntax is important and can be used anywhere within a deck. All declared variables are global in nature. Variables are case sensi-

tive and the operations involving them are done with the help of the “\$” character. The normal “\$” character can be referred to using the syntax “\$\$”. Variable names consist of an ASCII letter or underscore followed by a zero or more letters, numbers or underscores. Any other characters are illegal and result in an error. The syntax used for variable operations is:

```
$Variable_name or $(Variable_name)
```

The keywords used for related variable operations are setvar, input and select. For example:

```
<setvar name="Car" value="Ferrari" />
```

USER INPUT

WML provides various elements to obtain user input. One major advantage is the language-provided ability to customize user-input elements for accepting data. This decreases the redundant task of validating common errors in user input.

- **Input:** Input element provides a text field for users to key in the data (input method varies, depending on the platform). For example:

```
<input name="id" type="text" value="Bill"/>
```

- **Select:** Select element provides a list of items. Both multichoice and single-choice list elements are supported by WML. The contents of the select element can be set using the option element.

```
<select name="LIST" iname="L" ivalue="1;2" multiple="true">
<option value="B">Bus</option>
<option value="C">Car</option>
<option value="T">Train</option>
</select>
```

- **Optgroup:** Optgroup element groups related option elements into a hierarchy. This hierarchy helps the user agent provide the layout and presentation. Various option elements can be set under an optgroup element. The display is similar to multiple select elements that are combined into one.

```
<select name="drivers" multiple="true" tabindex="3">
<optgroup title="maclaren">
<option value="mika">Mika</option>
<option value="david">David</option>
</optgroup>
<optgroup title="williams">
<option value="ral">Ralph</option>
<option value="but">Button</option>
</optgroup>
</select>
```

- **Fieldset:** The fieldset element groups related elements of various types. This helps categorize fields on the basis of the application rules.

```
<fieldset title="Employee">
First name: <input type="text" name="fname"maxlength="25"/><br/>
Last name: <input type="text" name="lname" maxlength="25"/><br/>
</fieldset>
<fieldset title="Profile">
<select name="type">
<option value="P">Permanent</option>
<option value="T">Consultant</option>
</select><br/>
</fieldset>
```

IMAGES

Because of the limitations of the WAP devices, normal JPEG and GIF images can't be displayed or stored on them. A new image type called the


WBMP format has been specified for WAP devices. It's a one-bit variant of the normal BMP format and supports black or white. This graphics file is limited: it can't be more than 150x150 pixels or 1,461 bytes. As the capabilities of the wireless devices increase, these limitations are bound to change. You can incorporate these graphics files in your WML deck by using the href element. The syntax is:

```

```

Conclusion

The WAP forum's efforts to standardize WML and WMLScript make it easy for content and service providers to focus on providing more services and functionality rather than spending time catering to disparate wireless technologies. The continuing WAE efforts will provide more functionality and a higher level of control over wireless devices using WML and associated technologies.

This overview of WML has described as many elements of the language as possible. Next we'll see how WMLScript can be used in tandem with WML for developing WAP applications. Additional code listings can be found on the **XML-JWeb** site. 

Acknowledgment

The WINWAP WML browser was used to display the Output Example image.

References

1. WAP Forum: www.wapforum.org
2. WAE Overview: www.wapforum.org
3. WAP for Mobile Business: www.nokia.com/corporate/wap/index.html
4. Mobile Internet: www.ericsson.com/WAP/
5. Nokia Wireless Application Protocol Toolkit: www.forum.nokia.com/main.html

AUTHOR BIO

Mahesh Chulet, a Java consultant for Pentafour International, has worked as a software engineer and Java developer for more than two years. A Sun-certified Java 2 programmer, he has a diploma in advanced computing from the Center for Development of Advanced Computing in India.



LISTING 1

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="ex1" title="hello">
<p align="center">
<b> Hello World! </b></p>
<do type="accept" name="two" label="Next Card">
<go href="#ex2"/>
</do>
</card>
<card id="ex2" title="Welcome">
<p align="center">
<b> Quick Guide to WML </b>
</p>
</card>
</wml>
```





Generating source code easily using an automated process

Using Database Meta Data and XSL

Data-driven development defies database descriptions daily....Try saying that fast enough, and you'll have an idea of the problems involved in database programming these days. It was much simpler when the industry was built around client/server development and client code talked directly to a database server. When you wanted to get data from a database, you embedded a SQL query into the application code and there was no middle tier involved. Of course, there were some major disadvantages with this approach, and we've gotten a bit smarter since then. Now we're usually building distributed applications, communicating through application servers and Web servers.

One technique that companies use to simplify the development effort is to map relational database tables onto business objects in an object-oriented language. This approach can make code easier to maintain, since the client code doesn't contain SQL statements but instead performs operations on business objects. The client code is thus cleaner and less dependent on the physical database schema. The difficulty, of course, is in converting between the physical structure of the database and the logic of the business objects. Some very good tools are available to do this, but what can you do when a customer won't pay for the tools, or the customer doesn't want to spend time and money to learn a new API?

My customers frequently ask for Java applications based on business objects, and the data is usually stored in a relational database. An object-oriented database might have made more sense in many cases, but it's usually a difficult sell when you're talking with relational database administrators! One solution is to write source code for objects based on

the database schema. This is simple enough, if the schema is not very complicated or only a few tables are involved.

When a hundred or more tables are involved, however, the problem is a logistical one. In other words, the data entry itself becomes a major task. But rather than hire an army of clerks, it's possible to generate source code for the business objects directly from the database using an automated process. In the rest of this article I'll show you how to generate source code for some simple business objects using XSLT. The process involves generating an XML document to represent the database schema, and then using an XSL stylesheet to generate source code for the Java classes.

Obtaining the Database Schema

The first step is to obtain the database schema. In Java you can use the DatabaseMetaData class, which is part of the Java Database Connectivity (JDBC) API. This class provides metadata about a relational database; you can use it to read the structure of all the

tables in the database, as in Listing 1.

Many other methods of the DatabaseMetaData class provide access to information such as foreign key relationships and global database properties. The getTables() method returns a result set that has a list of all the tables, columns and data types in the database. The elements of this result set are shown in Figure 1, from the DatabaseMetaData documentation.

The result set contains very useful information about the database, but there is one extra step before you can convert the data into a usable XML file containing the database structure. The result set contains information about all the tables, but you'll need to group the information first before writing it to an XML document. If you tried to write this result set directly into an XML structure, it would look something like Listing 2.

But in order to loop through the items in each table, the document needs a hierarchical structure rather than a flat one such as the result set data. This is unfortunate, because XSL is particularly bad at grouping elements by a unique value. So you're much better off doing the grouping before you create the XML. The pseudocode in Listing 3 will group the results and build an XML tree using the Document Object Model (DOM).

After processing, the resulting document should look something like Listing 4.

Creating XSL Templates for Source Code

Now you have the database schema in an XML document that can be converted into a low-level database object. It's possible to do the conversion with XSLT. XSLT is designed primarily for converting XML documents into text formats such as HTML or another XML document. What many people don't realize is that XSL can be used to create other types of text documents, such as reports or, in this case, Java source code.

First I'll create a simple Java class that represents a row of data in a specific database table, as in the following code:

```
class People {  
    public String firstname;  
    public String lastname;  
    public java.util.Date DOB;  
}
```

To generate this source code from an XSL template, use the code in Listing 5.

The outer `<xsl:for-each>` element loops through all the table elements in the document, creating a class definition for each of them. The first `<xsl:value-of>` element extracts the name attribute of each matching table element and uses it as a class name. The inner `<xsl:for-each>` loop processes each column element within the table, using the column name attribute as a class field name and the

type attribute as the field's type. The `<xsl:text>` element is used to insert whitespace into the document.

The result of processing this template will be a file with a number of class definitions in it. You'll need to split this into separate files for each class if you need the classes to be public, since Java requires that all public classes be placed in their own source file. Once you compile the source files, you have Java class files to represent each table in your database. It's a lot easier than manually typing in each of the database column names to write the code. For the simple example given here, it may not be a huge time-saver, but the code could be extended further for more complex objects. If you wanted to make the objects into JavaBeans, for instance, you could modify the template to provide get and set methods for each field. Listing 6 gives the desired class. The XSL code to produce it appears in Listing 7.

Each class should now be roughly in a JavaBean format, although the method names don't follow the standard naming scheme. To fix that you'd need to change the capitalization for each method (see the JavaBeans specification at <http://java.sun.com>).

It's a bit of work to do the conversion in XSL, so I'll leave that for your homework!

For this simple example we've left the names unchanged.

Extensions and Limitations

This process can be further extended to generate more complex objects such as remote objects that could be used in an application server. Using some of the other methods of JDBC DatabaseMetaData, you can also get information on cross-references and primary keys in the database, and generate source code based on this information.

Some issues may need to be ironed out before you can use this process in your own application. For example, column and table names may conflict with existing classes and field names or with reserved words in Java. In those cases you can prefix names with character data that will prevent the naming conflict. Another problem with using XSL to generate source code is that you must escape certain characters, such as ampersand, less-than and greater-than symbols wherever they appear in the target source code. You must use the following XML entity references for this purpose:

Ampersand: `&`
Less-than: `<`
Greater-than: `>`

XSL has certain inherent limitations that may make it difficult to generate complex source code using only XSL. One of these limitations is the lack of grouping operations, where the data is grouped based on some unique value. In cases where that is required, it might be more appropriate to use an API such as SAX or DOM to parse the XML input, and to generate the source code from a Java program instead of XSL. One final issue is the complexity of large XSL files and the need to split source code into separate files before compiling public classes. And, of course, XSL with embedded Java source is a bit hard to read!

You'll also need some utility classes that move data into and out of the database and convert between SQL and business objects, and perform database operations (e.g., INSERT, SELECT, UPDATE and DELETE). These utilities aren't difficult to write, and can be reused in other applications once they're built. The basics of creating a framework for business objects, by reverse-engineering a database schema, are easy enough. With a little planning and practice you may be able to save yourself a lot of time! ☛

Only column descriptions matching the catalog, schema, table and column name criteria are returned.
They are ordered by TABLE_SCHEM, TABLE_NAME and ORDINAL_POSITION.

Each column description has the following columns:

1. TABLE_CAT String => table catalog (may be null)
2. TABLE_SCHEM String => table schema (may be null)
3. TABLE_NAME String => table name
4. COLUMN_NAME String => column name
5. DATA_TYPE short => SQL type from java.sql.Types
6. TYPE_NAME String => Data source dependent type name, for a UDT the type name is fully qualified
7. COLUMN_SIZE int => column size. For char or date types this is the maximum number of characters, for numeric or decimal types this is precision.
8. BUFFER_LENGTH is not used.
9. DECIMAL_DIGITS int => the number of fractional digits
10. NUM_PREC_RADIX int => Radix (typically either 10 or 2)
11. NULLABLE int => is NULL allowed?
columnNoNulls - might not allow NULL values
columnNullable - definitely allows NULL values
columnNullableUnknown - nullability unknown
12. REMARKS String => comment describing column (may be null)
13. COLUMN_DEF String => default value (may be null)
14. SQL_DATA_TYPE int => unused
15. SQL_DATETIME_SUB int => unused
16. CHAR_OCTET_LENGTH int => for char types the maximum number of bytes in the column
17. ORDINAL_POSITION int => index of column in table (starting at 1)
18. IS_NULLABLE String => "NO" means column definitely does not allow NULL values;

"YES" means the column might allow NULL values. An empty string means nobody knows.

FIGURE 1 Structure of getTables() result set

AUTHOR BIO

Brian Eubanks is the founder of Eu Technologies, Inc., a consulting and training firm based in Northern Virginia. Eu Technologies currently provides Java and XML consulting and training services to clients in the Mid-Atlantic region. Recent clients have included the New York Stock Exchange, government agencies, and public and private firms. Brian holds a master of science degree in computer science from George Mason University.

BRIAN@EUBOT.COM

LISTING 1

```
// get a database connection
Connection conn =
    DriverManager.getConnection(dbURL, username, password);
// get its metadata object
DatabaseMetaData dmd = conn.getMetaData();
// get a list of the tables and columns
// passing null parameters means no search criteria will be
// used
ResultSet rs = dmd.getTables(null,null,null,null);
```

LISTING 2

```
<column table="People" name="firstname" type="String" />
<column table="People" name="lastname" type="String" />
<column table="People" name="DOB" type="Date" />
<column table="Companies" name="companyname" type="String" />
<column table="Companies" name="address" type="String"/>
```

LISTING 3

```
// create the XML document
XmlDocument document = new XmlDocument();
// create the root database node
Element db = document.createElement("database");
// add it to document
document.add(db);
// create table node
Element table = null;
String previousTable = "";
while (rs.next()) { // for each row of result set
    String tableName = rs.getString("TABLE_NAME");
    if (!tableName.equals(previousTable)) {
        table = document.createElement("table");
        // set table name attribute
        table.setAttribute("name",tableName);
        // add table node to database node
        db.add(table);
    }
    // create the column node
    Element column = document.createElement("column");
    // set column attributes [name, type] from result set
    column.setAttribute("name", rs.getString("COLUMN_NAME"));
    // the data type is the SQL type, not the Java type,
    // and conversion will be required
    // (we've assumed an SQLToJava method is defined some-
    // where)
    String className = SQLToJava(rs.getString("DATA_TYPE"));
    column.setAttribute("type", className);
    table.add(column); // add column node to table node
    previousTable = tableName;
}
}
```

LISTING 4

```
<database>
<table name="People">
    <column name="firstname" type="String"/>
    <column name="lastname" type="String"/>
    <column name="DOB" type="java.util.Date"/>
</table>
<table name="Companies">
    <column name="companyname" type="String"/>
    <column name="address" type="String"/>
</table>
</database>
```

LISTING 5

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Trans-
```

```
form"
    version="1.0">
<xsl:output method="text" />
<xsl:template match="/database">
<xsl:for-each select="table">
class <xsl:value-of select="@name" /> {
    <xsl:for-each select="column">
        public <xsl:value-of select="@type" /><xsl:text>
</xsl:text><xsl:value-of select="@name" /> ;
    </xsl:for-each>
    }
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

LISTING 6

```
public class People {
    private String firstname;
    public String getfirstname() { return firstname; }
    public void setfirstname(String firstname) {
        this.firstname = firstname;
    }

    private String lastname;
    public String getlastname() { return lastname; }
    public void setlastname(String lastname) {
        this.lastname = lastname;
    }

    private java.util.Date DOB;
    public java.util.Date getDOB() { return DOB; }
    public void setDOB(java.util.Date DOB) {
        this.DOB = DOB;
    }
}
```

LISTING 7

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Trans-
form"
    version="1.0">
<xsl:output method="text" />
<xsl:template match="/database">
<xsl:for-each select="table">
    public class <xsl:value-of select="@name" /> {
<xsl:for-each select="column">
        private <xsl:value-of select="@type"/><xsl:text>
</xsl:text><xsl:value-of select="@name"/>;

        public <xsl:value-of select="@type"/> get<xsl:value-of
select="@name"/>() {
            return <xsl:value-of select="@name"/>;
        }

        public void set<xsl:value-of select="@name"/>
            (<xsl:value-of select="@type"/><xsl:text>
</xsl:text><xsl:value-of select="@name"/>) {
            this.<xsl:value-of select="@name"/> = <xsl:value-of
select="@name"/>;
        }
    }
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```



65 million digital cell phone users

4 million WAP phone enthusiasts

150,000 Palm VII fanatics

50,000 RIM pager loyalists

6,000 operating companies

200 carriers

11 networks

1 API



With Air2Web's Always Interactive™ API, write one XML-based application and reach the entire digital wireless market. Air2Web automatically optimizes your application for maximum interactivity on every device—supporting 2-way text, interactive audio, text-to-speech, voice recognition, and soon, video. Air2Web even provides customizable modules for common wireless functions, such as device authentication. Our hosted development environment makes it easy to define, create, test and deploy interactive applications that promote your brand. And as a wireless ASP, we host and support your applications in a reliable and secure environment. No hardware or software to buy. No carrier gateways to manage. No messaging protocols to understand. No wireless experience necessary. See us at Internet World, Booth #2035. Visit us at www.air2web.com. 404.815.7707

Always Interactive.

Loose coupling protocol provides ease in code change

An Approach to Efficient Information Transfer

XML provides a convenient mechanism for information exchange between heterogeneous applications. Information providers expose data that is of some value to their clients in the form of an XML document. The way information is processed varies from one client to another and is dictated by the actual application run by the client. A financial information provider, for example, runs an XML server application that provides company information (see Listing 1) to its clients, as shown in Figure 1. This information can be used by a Stock Quote application that tells its subscribers about the current stock price for a given ticker symbol, or by a stock analysis application that can graphically depict the performance of a stock over a period of time. It can also be used to generate a list comprising a company name, address and key employees for a directory service.

The Stock Quote application may be an intelligent Internet agent (see Figure 1) that provides services to a wide range of subscribers who may use a desktop computer or a wireless device to sign in. The agent serves the information in the format accepted by the device (i.e. a WAP phone subscriber is sent the information as WML, while other PDA users may receive the same information as HTML).

A Stock Quote application is interested only in the current stock price. It doesn't have any use for the performance history and company information that comes along in the XML document from the information provider. Likewise, Stock Analysis and Directory Service applications are interested only in a portion of the XML document and have no use for the extra baggage. This implies that a lot of bandwidth is wasted in transporting unused information.

A straightforward approach would make the XML server aware of the clients it's serving. That way, the server could generate an appropriate XML document for a given client. However, with this approach the server application is tightly integrated with the clients. The server needs to be modified with each new client application. Maintaining such a server can become a daunting

task and can also impact the availability of the server.

A better approach is to allow a client to tell the server what information it's interested in during the registration process. The client sends a template XML document when it registers with the server. When the client requests information, the server can filter the

XML document before sending it to the client. This way, the client controls what information it can expect from the server and the server is immune to change caused by new client applications. The template describing the information required by a Stock Quote application can be represented by `stockquote.xml` (see Listing 2).

Now I'll describe how to build an XML filter that can be used by the XML server, along with a simple protocol for registration and subsequent information exchange.

XML Filter

The XML filter is written in Java using the SAX API. SAX is an event-driven API

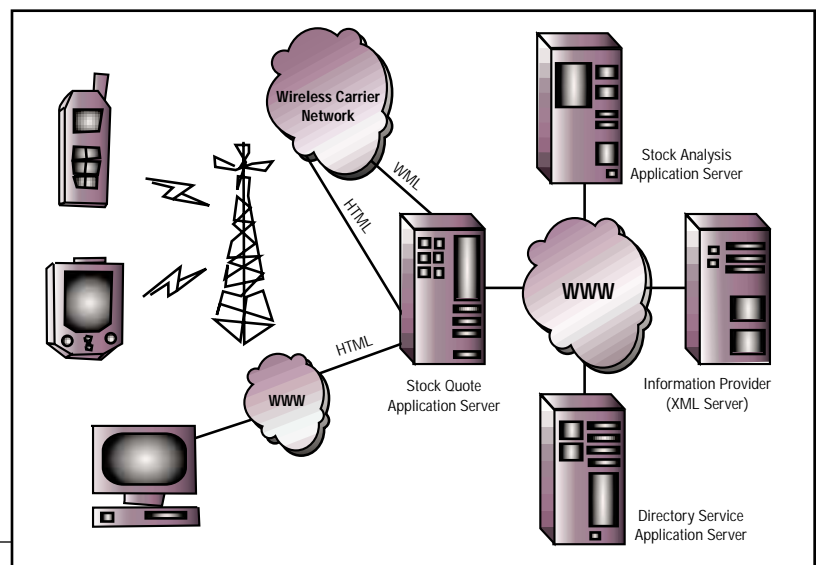


FIGURE 1 Information provider and client applications

AUTHOR BIO

Sameer Bhatia is a software consultant with 10 years of experience in distributed computing architectures using object-oriented design methodologies. He is also a Sun-certified Java programmer.

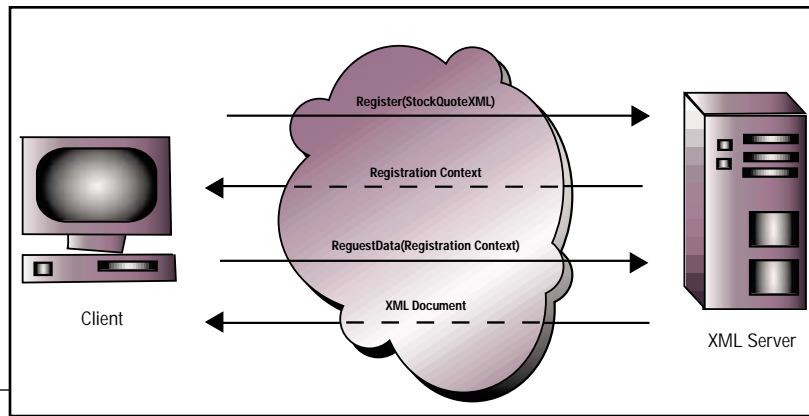


FIGURE 2 Registration and information exchange sequence

for parsing XML documents. The parser notifies the filter when certain events happen. The events that are used by the XML filter are:

- **startDocument:** Signals the start of the document.
- **startElement:** Signals the start of an element. The parser fires this event when the contents of the opening tag have been processed. This includes the name of the tag and any attributes it might have.
- **endElement:** Signals the end of an element.
- **characters:** Contains character data that is the actual value associated with a tag.

The sample code presented here uses the XML4J parser available from IBM. However, the code isn't dependent on any particular parser implementation as it strictly conforms to the SAX API as recommended by the W3C specification.

XML filtering is done in two phases. In the first phase an XMLFilterDescriptor (see Listing 3) object is created using the XML document supplied by the client. XMLFilterDescriptor parses the XML document using SAX API. The startElement event handler converts the

XML tags into fully qualified tag strings and stores them. A fully qualified tag string is a concatenated string of the tags that have been encountered from the document root through the current tag. A fully qualified tag string is required to differentiate between tags that have the same name in an XML document but a different meaning. For example, as shown in Listing 1, <high> included within the <january> tag is different from <high> within the <february> tag. A fully qualified tag name will appear as .companyList.company.performance.january.high.

In the second phase an XMLFilter (see Listing 4) object is created that parses the server XML document. The startElement event handler converts the XML start tag to a fully qualified tag string. This tag string is then searched in the XMLFilterDescriptor object, and if found, indicates a tag that is of interest to the client. All the associated attributes and their values are then sent to the client after formatting them as XML. The actual value associated with the tag, if any, is sent to the client by the Characters event handler. An end tag is sent to the client by the endElement event handler.

Registration and Information Exchange Protocol

Figure 2 shows the simple protocol used by the client and XML server. The client registers with the XML server specifying the information of interest. This information is an XML document that is actually a subset of the XML document available from the XML server. The Stock Quote application uses stockquote.xml (see Listing 2) to register with the information provider. The client receives a registration context after a successful handshake with the server.

The client makes a request to the XML server for information specifying the registration context. The XML server fetches the XMLFilterDescriptor object corresponding to the registration context, then filters the XML document using the XMLFilter object. The filtered XML document is then sent back to the client.

The code presented here shows how an HTTP client (see Listing 5) may communicate with a Java servlet-based server (see Listing 6) using a tunneling mechanism.

(In an actual implementation for a Stock Quote application it may be a better approach for the XML server to call back the client whenever it receives a stock update. This eliminates the periodic data requests the client application makes to the XML server.)

Conclusion

Without a doubt, using an XML filter provides a loose coupling between the XML server and its clients. New clients may come and older clients may go without requiring the server for a code change. This is particularly helpful in a business-to-business environment where new alliances are continuously formed for increased market presence. ☛

SBHATIA@GLOBALNOISE.COM

LISTING 1

```
company.xml
<?xml version="1.0" encoding="UTF-8"?>
<companyList>
  <company ticker="XYZ">
    <name>Nextgen Aerodynamics Inc</name>
    <address>2355 Lakeside Pkwy Alpharetta GA</address>
    <phone>(678)393-9090</phone>
    <officer>John B. England</officer>
    <officer>Cliff Anderson</officer>
    <officer>Bill Graham</officer>
    <officer>Roger Smith</officer>
    <quote>
      <lasttrade>
        <time>11:40 AM</time>
        <price>123.25</price>
      </lasttrade>
      <change>10.75</change>
```

```
<prevclose>130.45</prevclose>
<volume>12,500</volume>
<open>130.00</open>
<bid>123.50</bid>
<ask>124.75</ask>
</quote>
<performance>
  <january>
    <open>120</open>
    <close>110</close>
    <high>130.50</high>
    <low>90.45</low>
    <volume>60,000</volume>
  </january>
  <february>
    <open>130</open>
    <close>120</close>
    <high>160.50</high>
    <low>80.45</low>
```

```

        </february>
    </performance>
</company>
</companyList>

```

LISTING 2

```

stockquote.xml
<?xml version="1.0"?>
<companyList>
    <company ticker=" ">
        <name/>
        <quote>
            <lasttrade>
                <price/>
            </lasttrade>
            <bid/>
            <ask/>
        </quote>
    </company>
</companyList>

```

LISTING 3

```

XMLFilterDescriptor.java
package XMLFilter;

import java.io.*;
import java.util.*;
import org.xml.sax.*;

public class XMLFilterDescriptor extends HandlerBase
{
    public XMLFilterDescriptor(String saxDriver)
    {
        _tags = new Hashtable();
        _tagStack = new Stack();
        _tagStack.push("");
        _saxDriver = saxDriver;
    }

    boolean parse(String xmlDocument)
    {
        try
        {
            Class c = Class.forName(_saxDriver);

            Parser parser = (Parser)c.newInstance();
            parser.setDocumentHandler(this);
            parser.parse(xmlDocument);
        }
        catch (Exception e)
        {
            return false;
        }

        return true;
    }

    boolean parse(InputStream is)
    {
        try
        {
            Class c = Class.forName(_saxDriver);

            Parser parser = (Parser)c.newInstance();
            parser.setDocumentHandler(this);
            parser.parse(new InputSource(is));
        }
        catch (Exception e)
        {
            return false;
        }

        return true;
    }

    public void startElement(String name, AttributeList amap)
    throws SAXException
    {
        String element = (String) _tagStack.peek();
        String newElement = element + "." + name;

```

```

        _tagStack.push(newElement);
        _tags.put(newElement, newElement);
    }

    public void endElement(String name) throws SAXException
    {
        _tagStack.pop();
    }

    Stack _tagStack; // Builds Fully Qualified Tag
    Hashtable _tags; // Stores Fully Qualified XML tags
    String _saxDriver;
}

```

LISTING 4

```

XMLFilter.java
package XMLFilter;

import java.io.*;
import java.net.URL;
import java.util.*;
import org.xml.sax.*;

public class XMLFilter extends HandlerBase
{
    public XMLFilter(String xmlDocument, XMLFilterDescriptor
    filterDesc, String saxDriver)
    {
        _xmlDocument = xmlDocument;
        _filterDesc = filterDesc;
        _saxDriver = saxDriver;
        _tagStack = new Stack();
        _tagStack.push("");
        _validTag = false;
    }

    void setPrintWriter(PrintWriter pw)
    {
        _pw = pw;
    }

    void execute() throws Exception
    {
        Class c = Class.forName(_saxDriver);
        Parser parser = (Parser)c.newInstance();
        parser.setDocumentHandler(this);
        parser.parse(_xmlDocument);
    }

    public void startDocument() throws SAXException
    {
        _pw.println("<?xml version=\"1.0\" encoding=\"UTF-
8\"?>");
    }

    public void startElement(String name, AttributeList amap)
    throws SAXException
    {
        String element = (String) _tagStack.peek();
        String newElement = element + "." + name;

        _tagStack.push(newElement);

        if (_filterDesc._tags.containsKey(newElement))
        {
            StringBuffer tag = new StringBuffer("<");
            tag.append(name);
            for (int i = 0; i < amap.getLength(); i++)
            {
                tag.append(" ");
                tag.append(amap.getName(i));
                tag.append("=");
                tag.append('\"');
                tag.append(amap.getValue(i));
                tag.append('\"');
            }
            tag.append(">");
            _pw.println(tag.toString());
            _validTag = true;
        }
        else
        {

```

```

        _validTag = false;
    }
}

public void endElement(String name) throws SAXException
{
    String endElement = (String) _tagStack.pop();
    if (_filterDesc._tags.containsKey(endElement))
    {
        _pw.println("</" + name + ">");
    }
}

public void characters(char[] ch, int start, int length)
throws SAXException
{
    if (_validTag)
    {
        _pw.println( new String(ch, start, length));
    }
}

private Stack                _tagStack;
private boolean              _validTag;
private String                _xmlDocument;
private XMLFilterDescriptor _filterDesc;
private PrintWriter          _pw;
private String                _saxDriver;
}

```

LISTING 5

XMLClient.java

package XMLFilter;

import java.io.*;
import java.net.*;

public class XMLClient
{

```

    public XMLClient(String url)
    {
        _url = url;
    }

```

```

    public static void main(String[] args)
    {
        XMLClient stockQuoteClient =
            new XMLClient("http://localhost:8080/servlet/XMLServ-
er");

```

```

        // Register With XML Server.
        String context = stockQuoteClient.register();

```

```

        // Request Data from XML Server.
        stockQuoteClient.requestData(context);
    }

```

```

    public String register()
    {
        URLConnection connection = null;

        try
        {
            URL url = new URL(_url + "?register");

            connection = url.openConnection();
            connection.setUseCaches(false);
            connection.setRequestProperty("CONTENT-TYPE", "applica-
tion/octet-stream");
            connection.setDoInput(true);
            connection.setDoOutput(true);

            PrintWriter pw = new PrintWriter(connection.getOutput-
Stream(), true);
            BufferedReader br = new BufferedReader(new FileRead-
er("stockquote.xml"));

            // Send the XML document.
            String xmlString;
            while((xmlString = br.readLine()) != null)
            {

```

```

                pw.println(xmlString);
            }

            pw.close();
            br.close();

            // Recv Registration Context.
            br = new BufferedReader(new InputStreamReader(connec-
tion.getInputStream()));
            String context = br.readLine();
            br.close();

```

```

            return context;
        }
        catch (Exception e)
        {
            System.out.println("Exception: " + e);
        }

```

```

        return null;
    }

```

public void requestData(String context)

```

{
    URLConnection connection = null;

```

```

    try
    {
        URL url = new URL(_url + "?context=" + context);

```

```

        connection = url.openConnection();
        connection.setUseCaches(false);
        connection.setRequestProperty("CONTENT-TYPE", "applica-
tion/octet-stream");
        connection.setDoInput(true);
        connection.setDoOutput(true);

```

```

        BufferedReader br = new BufferedReader(new InputStream-
Reader(connection.getInputStream()));
        PrintWriter pw = new PrintWriter(System.out, true);

```

```

        // Recv XML Document.
        String xmlString;
        while((xmlString = br.readLine()) != null)
        {
            pw.println(xmlString);
        }

```

```

        pw.close();
        br.close();
    }

```

```

    catch (Exception e)
    {
        System.out.println("Exception: " + e);
    }
}

```

String _url; // URL of XML Server.

LISTING 6

XMLServer.java

package XMLFilter;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class XMLServer extends HttpServlet
{

```

    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);

```

```

        _contextTable      = new Hashtable();
        _saxDriver           = "com.ibm.xml.parser.SAXDriver";
        _clientCounter      = 1;
        _xmlDocument        = "company.xml";
    }

```

Go Online and Subscribe Today!

www.SYS-CON.com



or

Call 1-800-513-7111

Subscribe to the Finest Technical
Journals in the Industry!

GET YOUR OWN!

```

public void service(HttpServletRequest request, HttpServlet-
tResponse response)
throws ServletException, IOException
{
    try
    {
        response.setContentType("text/html");

        String context = null;

        if (request.getParameter("register") != null)
        {
            registerClient(request, response);
        }
        else if ((context = request.getParameter("context"))
!= null)
        {
            sendXMLDocument(context, request, response);
        }
    }
    catch(Exception e)
    {
    }
}

synchronized void registerClient(HttpServletRequest request,
HttpServletResponse response)
throws Exception
{
    // Generate a Unique Context
    String context = "Context" + _clientCounter++;

    // Generate an XMLFilterDescriptor
    // and associate with Context
    XMLFilterDescriptor filterDesc = new XMLFilterDescrip-
tor(_saxDriver);
    if (filterDesc.parse(request.getInputStream()))
    {
        _contextTable.put(context, filterDesc);
    }

    // Send back the Context
    PrintWriter pw = new PrintWriter(response.getOutput-
Stream(), true);
    pw.println(context);
}

void sendXMLDocument(String context, HttpServletRequest
request, HttpServletResponse response)
throws Exception
{
    // Get XMLFilterDescriptor associated with Context
    XMLFilterDescriptor filterDesc = (XMLFilterDescriptor)
_contextTable.get(context);
    if (filterDesc == null)
    {
        return;
    }

    // Filter the XML Document using XMLFilterDescriptor
    XMLFilter xmlFilter = new XMLFilter(_xmlDocument, fil-
terDesc, _saxDriver);
    xmlFilter.setPrintWriter(new PrintWriter(response.getOut-
putStream(), true));
    xmlFilter.execute();
}

private Hashtable _contextTable;
private static int _clientCounter;
private String _saxDriver;
private String _xmlDocument;
}

```



NetDive

www.netdive.com

[WRITTEN BY JAMES A. BRANNAN]

Using XML with XSL doesn't mean you have to eschew your favorite server-sided scripting language for XML/XSL. XML data islands offer you the best of both worlds.

XML data islands are a powerful technique for embedding XML data in HTML documents. Internet Explorer 5 (IE5) is currently the only browser supporting them. However, you can still enjoy the flexibility of XML data islands if you process the XML on your server and insert the resulting HTML into your server page. I'll demonstrate this technique using Active Server Pages (ASP) and a C++ Active Template Library (ATL) component. I'll assume the reader has a basic understanding of XML, XSL and Microsoft's XML parser. I'll explain the C++ component in general terms that should be understandable to non-C++ programmers. Although I'm using Microsoft technologies, the concept of XML data islands is relevant regardless of the platform.

As the complexity, importance and size of Web applications increase, design issues become increasingly critical. A dominant strategy is a three-tiered architecture – dividing the application into display, business and database layers. Typically the display layer consists of cascading stylesheets, HTML documents and dynamic scripting pages such as ASP, JavaServer Pages (JSP) and ColdFusion. The business layer consists of server-sided scripting pages and, frequently, components such as COM objects or Java servlets. The database layer consists of the database, associated stored procedures and database-related components.

Three-tiered strategies often break down in practice. A single server-sided scripting page or component often combines display logic with both business and data logic. For example, an ASP page that gets and transforms data, then formats its display – all in one page – is all too common. This combination of logic (a two-tiered design) is often problematic. It's usually harder to debug and distinguish what the different code is doing in a two-tiered application. These applications are also not as scalable and require increased regression testing every time the code changes.

XML and XSL address this problem. Display logic is contained in XSL stylesheets. Business and database logic are contained in components and/or server-sided scripting pages. The business logic is responsible for getting XML, performing the necessary transformations and applying an XSL stylesheet to the XML data. This separation results in applications that are easier to debug, modify and maintain.

XML data islands are XML data embedded in HTML documents. IE5 can process an XML data island directly. JavaScript or VBScript combine the XML and XSL stylesheets using IE5's XML parser, MSXML. We can also use MSXML on the server independent of IE5. After processing the XML data on the server, we insert the area of data-driven HTML into an ASP page. Using MSXML on the server enables us to use an ASP page while using XML with XSL. The result of the XSL transformation on the XML data isn't a complete HTML document, but rather an HTML document fragment.

Consider a hypothetical Internet application for viewing a list of all employees. This list needs to be formatted output for viewing in Web browsers and handheld devices, as well as raw data so future applications can obtain it easily using HTTP. XML combined with XSL meets these requirements. Obtaining different display formats is as easy as

FLEXIBLE
XML WEB
DESIGN:
XML
DATA ISLANDS

Combining server-side scripting languages with the power of XML and XSL

applying different XSL stylesheets to the XML data. Building a switch that causes the ASP page to return XML only meets the requirement to expose the raw data to other applications.

Architectural Overview

The class diagram in Figure 1 provides a logical overview of our site and a better understanding of the components it is composed of. The notation is Unified Modeling Language (UML) with Web application extensions. The pages DeptListings and EmpDisplay_1/EmpDisplay_2 are displayed by the browser. EmpDisplay_1 and EmpDisplay_2 are actually the same page as EmpListing, but we model them separately due to the dynamic nature of server-sided scripting pages. These pages have two lives. Their first is on the server (EmpListing) and their second is in the client's browser (EmpDisplay_1 and EmpDisplay_2). The further distinction between EmpDisplay_1 and EmpDisplay_2 illustrates that for IE5, EmpDisplay_2 needs to contain two <xml> tags, a <div> tag and a large block of JavaScript to transform the XML (transform.js).

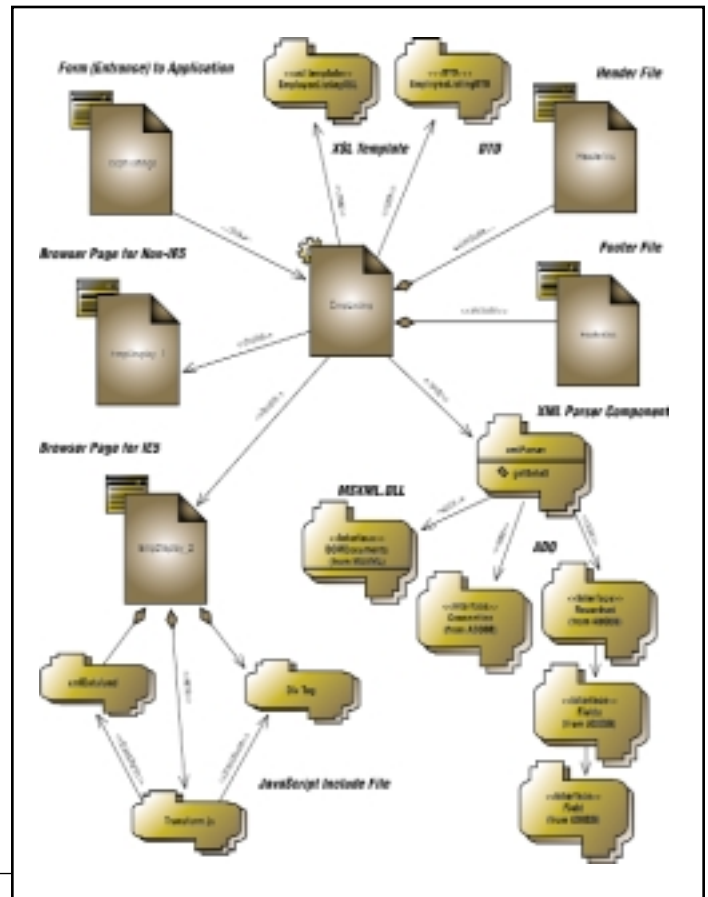


FIGURE 1 Class diagram of our application

Dynamic Overview

The sequence diagram in Figure 2 illustrates the sequence of messages in our application; it provides a high-level overview of the messages passed among the different elements. As the diagram illustrates, first DeptListings calls EmpListing; then, with the help of xmlparser, EmpListing is built. It's built differently depending on whether XML or HTML was requested. Then it's sent to the browser. We break down EmpListing's processing in the following steps:

1. Dimension any variables needed and instantiate the xmlparser component.
2. Determine if the URL parameter indicates rawxml or html.
3. If rawxml, call xmlparser.getData and output the results (XML).
4. If not rawxml, determine if the browser is IE5.
5. If not rawxml and not IE5:

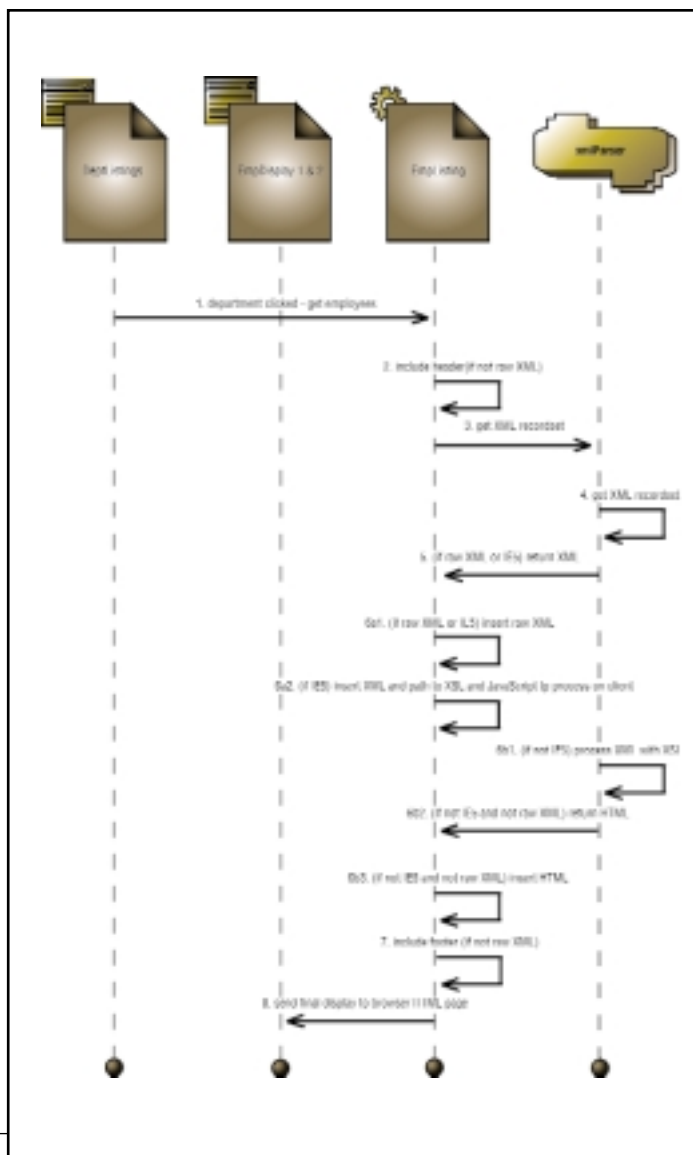


FIGURE 2 Sequence diagram of our site

- Include the header
- Call `xmlparser.getData`, making sure to pass the XSL template and the DTD (Document Type Definition) path, and write the processed results to the browser.
- Include the footer.
- 6. If not `rawxml` and IE5:
 - Write the JavaScript needed to process the XML data island to the browser.
 - Write a body tag with an `onLoad` method that calls the JavaScript function.
 - Include the header.
 - Call `xmlparser.getData` and place the results in a variable in `EmpListing`.
 - Place the opening tag `<xml id="xmldata">` and closing tag `</xml>` around the XML.
 - Add the opening tag `<xml id="xsldata" src="ourpath">` and closing tag to the end of the variable.
 - Write the variable to the browser.
 - Include the footer.

Essentially, if the user isn't using IE5, get HTML from the XML component; otherwise put both the XML data and the XSL template in `<xml>` tags and include a JavaScript function. This function uses the two `<xml>`

tags to transform the data and display it in a `<div>` tag using the `<div>` tag's `innerHTML` property.

There are several techniques for transforming data from a database into XML. If using ADO 2.1, we can use the `save` method of a recordset and persist it as XML. If using ADO2.5, we can write the ADO output directly to the output stream as XML. However, both solutions tie us to Microsoft's view of how XML data should appear (Microsoft's XML Reduced Data Schema Format). Two more general techniques are: (1) build the XML DOMDocument node by node or (2) create a large XML string and load it into an XML DOMDocument.

There are benefits to both solutions. In the first one we instantiate an XML document and build it. Building the document directly gives us access to all the XML DOMDocument's methods. However, instantiating the XML DOMDocument object and all its subobjects can be costly. As reported on MSDN by Chris Lovett, program manager for Microsoft's XML team, Microsoft's own tests demonstrate that loading a string is "roughly five times faster" than building a document node by node. In our design, if the user has IE5 there's no reason to instantiate a DOMDocument on our server at all. Only when the client requests formatted output and doesn't have IE5 do we instantiate an XML DOMDocument, load the XML string and transform it. For these performance reasons we chose the later technique.

The `xmlparser` component gets the XML and transforms it. This process is summarized in the following steps:

1. Declare and initialize any needed variables.
2. Get the recordset (for example, ADO).
3. Allocate enough space for a large data string (the XML).
4. Begin the XML string (including a reference to the DTD).
5. Repeat for each record:
 - Append record opening tag to XML string.
 - Repeat for each field:
 - Append field opening tag to XML string.
 - Append field data to XML string.
 - Append field closing tag to XML string.
 - Append record closing tag to XML string.
6. Close the XML string.
7. Determine if XML or HTML is to be returned.
8. If XML, return the XML string.
9. If HTML:
 - Instantiate the XML/XSL parser (for example, `MSXML.DLL`).
 - Ensure the XML parser is set to validate the XML with our DTD.
 - Load the XML.
 - Load the XSL template.
 - Process the XML with the XSL.
 - Return the transformed data.

The server-sided scripting page is the "glue" that holds our application together. It inserts any other files needed (such as headers and footers), instantiates our component, passes the required parameters to it, then inserts the output. The component that transforms the ADO to XML and the XSL stylesheet does most of the work.

Implementation

In this example we use ASP, a C++ ATL COM component and Microsoft's XML parser, `MSXML`. Our C++ component uses ADO and `MSXML` via the

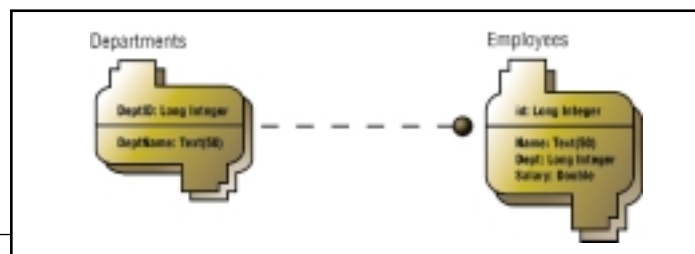


FIGURE 3 The site's database

#import statements (see Listing 1). Our data is from a simple two-table Access database (see Figure 3) with an ODBC connection named *employees*.

XML Component Implementation

Neither scripting languages nor Visual Basic handle string concatenation very efficiently. However, a Visual C++ ATL component that preallocates the space needed for the XML string will be fast. We implement *xmlparser* using C++. Listing 2 shows our component's primary method. The ATL wizard-generated IDL code isn't listed. Our method's signature is:

```
getData(BSTR *pConnection, BSTR *pQuery, BSTR *pDTDPath, BSTR
*pXslTemplatePath,
BSTR *pTransformedData)
```

Note that **pTransformedData* is the return data, not an input parameter. Our method takes a connection to an ODBC data source, the query, and the paths to the XML DTD and the XSL stylesheet.

We implement the code in Listing 2 as envisioned earlier. Even without a solid understanding of C++, you should be able to get the gist of what the method is doing: getting an ADO recordset, looping through that recordset and creating an XML string (using the string template in the Standard Template Library). If the method returns XML, the line:

```
*pTransformedData = SysAllocString(wstrXmlString.c_str());
```

is executed and XML is returned. If the method returns HTML, the string and the XSL template are loaded into XML DOMDocument objects, the XSL transforms the XML and HTML is returned. Since we're using a DTD, we turn on validation with the code:

```
SpXmlDocument->put_validateOnParse(true)
```

The error-handling skeleton code is included but not implemented.

Our use of MSXML is "low-rent." Although MSXML has a rich object hierarchy, we used only two: the top-level object (DOMDocument) and the *parseError* object. We needed to use only six methods. The method *put_validateOnParse* ensures the parser validates the XML with the DTD. The XML string is loaded via the *LoadXML* method. The XSL stylesheet is loaded using its path and the *Load* method. The code also uses the *get_parseError* and *get_errorCode* to check for any errors after the XML and XSL are finished loading. If there are no errors, the line:

```
*pTransformedData = SysAllocString(spXmlDocument->
transformNode(spXslDocument))
```

uses the DOMDocument containing the XSL, transforms the XML to the desired output and writes it to *pTransformedData*.

Active Server Page

Listing 3 contains the implementation of *DeptListing*, and Listing 4 contains the implementation of *EmpListing*. First, any needed variables are dimensioned and we instantiate our XML processing component (*xmlparser.dll*). If we need only XML returned to the browser, then the processing consists of:

```
Response.Write(xmlProcessor.getData("employees", cstr(strQuery),
"Listing.dtd", "rawxml"))
```

where *employees* is the DSN, *strQuery* is the SQL query statement, *listing.dtd* is the DTD and *rawxml* indicates we want XML returned. If the user wants XML we're finished.

If the user wants HTML, we must do more processing. First, determine if the user's browser is IE5. If it is, use the following code to write the necessary data, since IE5 can process XML data islands directly:

```
<%if instr(Request.ServerVariables("HTTP_USER_AGENT"), "MSIE 5.") >
0 then %>
```

```
<SCRIPT LANGUAGE=javascript src="transform.js">
</SCRIPT>
<body onLoad="transformXml();">
```

and

```
<div id="xslTarget"></div>
<xml id="xmldata">
<%
Response.write(xmlProcessor.getData("employees", cstr(strQuery),
"Listing.dtd",
"rawxml"))
%>
</xml>
<xml id="xsldata" src="listings.xsl"></xml>
```

I'll discuss the JavaScript page, *transform.js*, later. For now, all we need to know is that it contains a function called *transformXML*, which converts the XML data contained in `<xml id="xmldata"></xmldata>` to HTML using the XSL stylesheet in `<xml id="xsldata" src="listings.xsl"></xml>`, and writes it to the innerHTML of the tag `<div id="xslTarget">`. Both the text between the `<xml>` tags and the text contained in the page indicated by the *src* attribute are treated as XML by IE5.

If the user's browser isn't IE5, the ASP page includes the header and the footer, maps a path to the XSL template and calls the *getData* method of *xmlparser*:

```
strXslTemplate = server.MapPath ("listings.xsl")
Response.write(xmlProcessor.getData("employees", cstr(strQuery),
cstr(server.mappath("Listing.dtd")), cstr(strXslTemplate)))
```

Rather than passing *rawxml* as a parameter, *EmpListing* passes the XSL stylesheet path to *xmlparser*. The component *xmlparser* transforms the XML to HTML and returns the results.

JavaScript for IE5

Listing 5 contains the code for *transform.js*. This code applies the XSL stylesheet to the XML data island (when the user's browser is IE5). Although I don't explain MSXML's error handling, I include the code needed to handle errors. The statement:

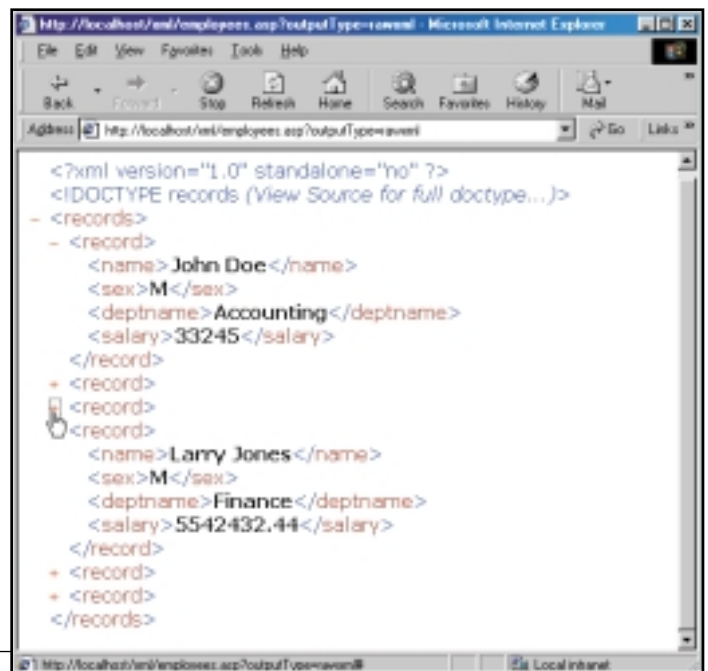


FIGURE 4 Resulting XML from our component

```
result = xmldata.transformNode(xmldata.XMLDocument);
```

transforms the XML with the XSL, and the statement:

```
xslTarget.innerHTML = result;
```

inserts the results into the <div> tag. Note that the try/catch block is IE5-specific JavaScript.

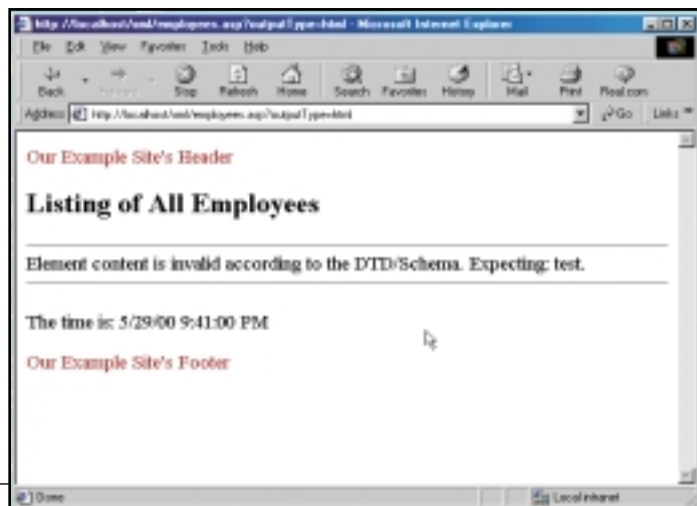


FIGURE 5 Error message when DTD doesn't match XML

XML Data and DTD

The resulting XML is shown in Figure 4. The root tag is <records>. Each ADO record becomes an XML <record> and each ADO field is a child element of <record>. Our application uses a DTD for a practical reason: specifying the DTD in Listing 6 ensures that MSXML catches changes to the database or query. The line:

```
<!ELEMENT record (name, sex, deptname, salary)>
```

ensures that each record contains a name, sex, deptname and salary child element. For example, as shown in Figure 5, the DTD code:

```
<!ELEMENT record (name, test, sex, deptname, salary)>
```

combined with an XML record that contains name, sex, deptname and salary (but not test) causes MSXML to throw an error because the XML doesn't have a <test> element and the DTD requires it. The DTD helps keep the configuration of the database/queries and the XSL templates in sync. Note that there's a performance penalty for validating an XML dataset.

XSL Stylesheet

The XSL stylesheet (see Listing 7) applies formatting rules to XML that result in the desired display. Our stylesheet contains multiple templates, indicated by the tag <xsl:template match="matchingnode">. Our main template, <xsl:template match="/">, matches our document's root element. Inside that template we call a second template when <records> is matched. Since the XML data set has one <records> tag, we apply the second template once. Inside the records template we loop through each record using the statement <xsl:for-each select="record">.

For each record we build a table row containing five table cells. Two of the cells contain data obtained from the XML data set using the <xsl:value-of> tag. The third cell applies a template to salary. This template contains an <xsl:eval> statement. The JavaScript that's inside the <xsl:eval> tag formats the node if it isn't blank:

```
<xsl:template match="salary">
  <xsl:eval>if (this.text != "") formatnumber(this.text,
    "#,##0.00");</xsl:eval>
</xsl:template>
```

Odd and even rows have a different background color. The code:

```
<xsl:script>
  <![CDATA[function even(e) {return absoluteChildNumber(e)%2==0;}]]>
</xsl:script>
```

and

```
<tr>
  <xsl:choose>
    <xsl:when expr="even(this)">
      <xsl:attribute name="bgcolor">#9999cc</xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
      <xsl:attribute name="bgcolor">#ccccff</xsl:attribute>
    </xsl:otherwise>
  </xsl:choose>
```

accomplishes this coloring. The function even returns true or false depending on whether the number is even. This function is called by the conditional statement <xsl:when expr="even(this)"> inside the <xsl:choose> block. The parameter "this" refers to the current node, <record>. The JavaScript function then uses the absoluteChildNumber property of the XML DOMDocument to determine if the current node is odd or even. The attribute bgcolor is added to the <tr> tag and the transformed tag becomes either:

```
<tr bgcolor="#9999cc">
```

or

```
<tr bgcolor="#ccccff">
```

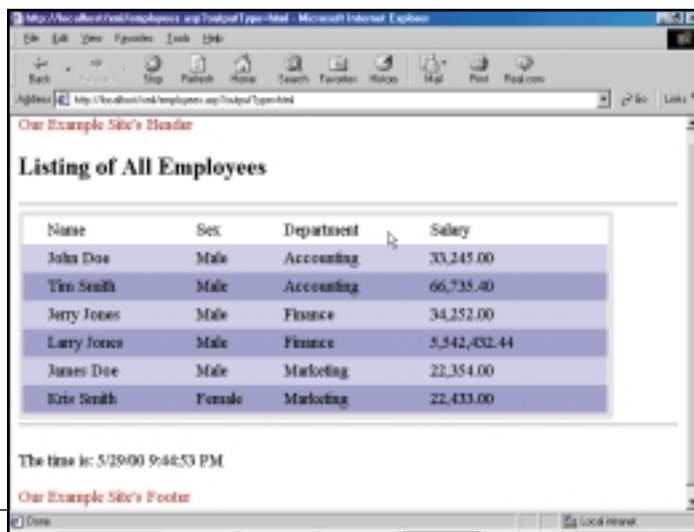


FIGURE 6 Result of XSL transformation of XML

Conclusion

The resulting output is illustrated in Figure 6. We breezed over quite a few concepts, and I didn't discuss how to program in C++ or write an XSL template. Rather, I focused on a practical example of what you can do with these technologies. Table 1 lists numerous online resources that expand on the architecture and technologies presented.

Data islands are a great way to implement XML in your site. They

ATL Components with Visual C++		
HOWTO: Create ATL COM Objects	http:suport.microsoft.com (Article ID: Q181265)	
Dr. GUI and ATL	www.msdn.microsoft.com	
Agility in Server Components	www.mdn.microsoft.com	Neil Allain
ADO in ATL Components		
Much ADO About Data Access	www.devx.com	Lyn Robinson
How Visual C++ Users Should Read the ADO Documentation	http:msdn.microsoft.com	
INFO: Using ActiveX Data Objects		
(ADO) via #import in VC++	http:support.icrosoft.com (Article ID: Q169496)	
ADO in your ATL Component	www.aspzone.com	John R. Lewis
MSXML.DLL Implemented with C++		
C++ Example Code	www.msdn.microsoft.com	
A Flying Start to Programming IE5/XML using C++ & COM	www.comdeveloper.com	Richard Anderson
General Architecture and XML/XSL Articles		
The Two-Tier Web Problem	msdn.microsoft.com	Jason Masterman
Construct Your E-Commerce Business Tier the Easy Way with XML, ASP, and Scripting	www.microsoft.com	Dave Cohen
Ten Best Bets for XML Applications	www-4.ibm.com	Bob Schloss
ArchitectureX: Designing for XML	www.xmlmag.com	Kurt Cagle
Introduction to DOM	www.developerlife.com	Nazmul Idris
XML in a Three-Tier Environment	www.developer.earthweb.com	Benoit Marchal
When Do You Adopt XML?	www.xmlmag.com	David S. Linthicum
Should I use SAX or DOM?	www.developerlife.com	Nazmul Idris
The Data Melting Pot: Building a Business-to-Business Application with XML	www.ibm.com	Brett McLaughlin
Modeling ASP Applications with UML	www.asptoday.com	Jim Conallen
XML		
Generating XML From ADO Recordsets	www.inquiry.com	Kurt Cagle
XML at Work: ColdFusion and Microsoft's XMLDOM Object	www.xml-journal.com	David Gassner
Which is Faster - Loading a DOM or Loading a String?	www.vbxml.com	
XSL		
XSL The Extensible Style Language	www.webtechniques.com	Norman Walsh
Chapter 14 of the XML Bible: XSL Transformations	http:metalab.unc.edu/xml/books/bible/updates/14.html	
Enhancing XSL	http:msdn.microsoft.com	Kurt Cagle
Transform Your Data with XSL	www.xmlmag.com	Kurt Cagle

TABLE 1 Online XML, XSL and ATL COM references

enable you to use server-sided scripting languages with the power of XML and XSL. The resulting output of XML and XSL is simply an HTML document fragment that's stuck in an ASP page. Localizing display logic to an XSL template avoids many of the difficulties associated with two-tiered Web design. Finally, allowing a page to have a "switch" that causes it to return only XML makes the page's data easily accessible to other applications. ☛

Reference

C++ ATL Component Tutorial: www.asptoday.com/articles/20000824.htm

AUTHOR BIO

James A. Brannan is a consultant specializing in Internet programming in the Washington, DC, metropolitan area. He is also pursuing a master's degree at the University of Maryland.

BRANNANJ@IEEE.ORG

LISTING 1

```
//include the string standard template library
#include <string>
using namespace std;
//import both ADO and XML parser
#import "c:\program files\common\system\ado\msado15.dll"
no_namespace rename("EOF", "adoEOF")
#import "c:\windows\system\msxml.dll"
rename_namespace("xml")
```

LISTING 2

```
STDMETHODIMP CxmlDataSet::getData(BSTR *pConnection, BSTR
*pQuery, BSTR *pDTPPath,
BSTR *pXslTemplatePath, BSTR *pTransformedData)
{
//declare the recordset pointer, connection pointer,
field pointer
_RecordsetPtr pRs;
_ConnectionPtr pCn;
FieldPtr spField;
_variant_t vtEmpty;
//declare an xml document smart pointer for the xml and
//the xsl stylesheet also declare an error pointer
xml::IXMLDOMDocumentPtr spXmlDocument;
xml::IXMLDOMDocumentPtr spXslDocument;
xml::IXMLDOMParseErrorPtr spXmlError;
//number of fields
long lngFieldCount;
//the iterator for the fields
long lngIndex = 0;
//variable for HRESULT
HRESULT hr = S_OK;
//string to hold the xml string being generated
basic_string<WCHAR> wstrXmlString;
//the connection DSN name
CComBSTR ccbstrConnection;
//the SQL query
CComBSTR ccbstrQuery;
//the xsl template path
CComBSTR ccbstrTemplate;
try
{
//set the variables
ccbstrConnection.Append(*pConnection);
ccbstrQuery.Append(*pQuery);
ccbstrTemplate.Append(*pXslTemplatePath);
//***** get the recordset *****
pCn.CreateInstance(_uuidof(Connection));
pRs.CreateInstance(_uuidof(Recordset));
pCn->Open(ccbstrConnection.m_str, L"", L"", -1);
pRs = pCn->Execute(ccbstrQuery.m_str, &vtEmpty,
adCmdUnknown);
lngFieldCount = pRs->Fields->GetCount();
//***** create the xml string *****
//reserve large block of memory for
//appending without reallocating memory
wstrXmlString.reserve(8500 * sizeof(WCHAR));
//begin the xml string
wstrXmlString.append(L"<?xml version='1.0' stand-
alone='no'>");
wstrXmlString.append(L"<!DOCTYPE records SYSTEM \"");
wstrXmlString.append(*pDTPPath);
wstrXmlString.append(L"\">");
wstrXmlString.append(L"<records>");
pRs->MoveFirst();
while(! (pRs->adoEOF))
{
wstrXmlString.append(L"<record>");
for (lngIndex = 0; lngIndex < lngFieldCount;
lngIndex++)
{
spField = pRs->Fields->GetItem(lngIndex);
wstrXmlString.append(L"<");
wstrXmlString.append(_wcslwr(spField->Get-
Name()));
```

LISTING 3

LISTING 5

```

function transformXml()
{
    var result;
    var errReason;
    var blnLoad = true;
    if (xmldata.parseError.errorCode != 0)
    {
        blnLoad = false;
        errorReason = xmldata.parseError.reason;
        xslTarget.innerHTML = errorReason;
        //other error handling code inserted here
    }
    else
    {
        if (xsldata.parseError.errorCode != 0)
        {
            blnLoad = false;
            errorReason = xsldata.parseError.reason;
            xslTarget.innerHTML = errorReason;
            //other error handling code inserted here
        }
        else
        {
            try
            {
                result =
xmldata.transformNode(xsldata.XMLDocument);
                blnLoad = true;
                //other error handling code inserted
here
            }
            catch(exception)
            {
                blnLoad = false;
                errorReason = xmldata.parseError.reason;
                xslTarget.innerHTML = errorReason;
                //error handling here
            }
        }
    }
    if (blnLoad == true)
    {
        xslTarget.innerHTML = result;
    }
}

```

LISTING 6

```

<?xml version="1.0"?>
<ELEMENT records (record+)>
<ELEMENT record (name, sex, deptname, salary)>
<ELEMENT name (#PCDATA)>
<ELEMENT sex (#PCDATA)>
<ELEMENT test (#PCDATA)>
<ELEMENT deptname (#PCDATA)>
<ELEMENT salary (#PCDATA)>

```

LISTING 7

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
xmlns:oursite="http://www.oursite.com">
<xsl:template match="/">
    <xsl:script>
        <![CDATA[function even(e) {return
absoluteChildNumber(e)%2==0;}}]>
    </xsl:script>
    <table width="90%" border="1">
    <tr>
    <td>
        <table width="100%" border="0" cellpadding="0" cell-
padding="5">
            <xsl:apply-templates select ="records"/>
        </table>

```

```

    </td>
    </tr>
    </table>
</xsl:template>
<xsl:template match="records">
    <tr>
    <td>
    <xsl:entity-ref name="nbsp"/>
    </td>
    <td>Name</td>
    <td>Sex</td>
    <td>Department</td>
    <td>Salary</td>
    <td>
    <xsl:entity-ref name="nbsp"/>
    </td>
    </tr>
    <xsl:for-each select="record">
        <tr>
        <xsl:choose>
            <xsl:when expr="even(this)">
                <xsl:attribute
name="bgcolor">#9999cc</xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
                <xsl:attribute
name="bgcolor">#ccccff</xsl:attribute>
            </xsl:otherwise>
            </xsl:choose>
            <td>
    <xsl:entity-ref name="nbsp"/>
    </td>
            <td>
    <xsl:value-of select="name"/>
    <xsl:entity-ref name="nbsp"/>
    </td>
            <td>
    <xsl:choose>
        <xsl:when test="sex[.='M']">
            Male
        </xsl:when>
        <xsl:otherwise>
            Female
        </xsl:otherwise>
    </xsl:choose>
    </td>
            <td>
    <xsl:value-of select="deptname"/>
    <xsl:entity-ref name="nbsp"/>
    </td>
            <td>
    <xsl:apply-templates select="salary"/>
    <xsl:entity-ref name="nbsp"/>
    </td>
            <td>
    <xsl:entity-ref name="nbsp"/>
    </td>
        </tr>
    </xsl:for-each>
</xsl:template>
<xsl:template match="salary">
    <xsl:eval>if (this.text != "") formatnumber(this.text,
"#,##0.00");</xsl:eval>
</xsl:template>
</xsl:stylesheet>

```





Parse XML data into C++ classes for increased performance

HighPerformance XMLParsinginC++

In my last article (*XML-J*, Vol. 1, issue 3) I made the case for using custom classes derived from XML Schemas to represent XML documents in C++ applications. That article focused primarily on the problems of generating XML documents from program objects, and explained how custom classes have significant advantages over standards like DOM and SAX in terms of performance, object orientation and maintainability of source code.

Here I'll describe a unique methodology for parsing XML data into C++ classes that provides all the object-oriented benefits detailed in the first article, with increased performance (compared to traditional generic XML parsers).

The Problem with Conventional Parsers

C++ programmers have been dealing with parsing technologies for years. Most of you remember writing simple language parsers in school, and probably wrote the basic syntax parser in tools like Lex and Yacc. So, for C++ developers, the idea of a syntax parser isn't especially intimidating.

The basic grammar for XML is pretty simple compared to a programming language like C++ or Java, for example, but there's one problem unique to XML parsing that *is* daunting: unlike conventional programming languages, XML doesn't have a fixed set of tags (i.e., keywords). Imagine trying to develop a general-purpose grammar for a programming language with a user-defined set of keywords!

To solve the general problem of XML parsing, it's necessary to build a parser that can be dynamically fed a list of tags and rules for the specific dialect of XML to be parsed. In the terminology of XML standards, that means specifying an XML Schema file to a DOM parser so that it knows how to parse and validate the specific dialect of the input XML file.

If an application reads and writes a variety of dialects of XML documents,

the DOM model is appropriate because it doesn't require source code changes for incremental support for a new dialect of XML. This is typically the case for integration broker applications, as described in my last article, in which the broker is reading, transforming and forwarding all kinds of XML documents within and between organizations.

However, as I also described, there's a large class of applications in which only a few types of XML are spoken and these don't often change. For these, the overhead of DOM and the lack of application-specific object orientation is a major drawback.

Static Parsers Derived from XML Schemas

Just as it's beneficial in some environments to derive C++ classes from XML Schemas for writing XML documents, it can also be beneficial to derive classes to read XML documents from schemas.

The typical process for creating a language parser in C++ is to hand-code the Lex rules and Yacc grammar, then generate the Lexer and parser from these XML

dialect-specific input files (see Figure 1).

This process is tedious, however, and must be redone for each dialect of XML that your application needs to parse. While doable, the same logic that you'd hand-code in the rules and grammar is already encapsulated in the XML Schema file. A more efficient approach is to develop a translation program that can convert the XML Schema file into the equivalent Lex rules and Yacc grammar for the XML dialect (see Figure 2).

The example project in Listing 1 shows a generated grammar for a sample XML DTD file called `acmecpc.dtd`. You'll see the generated Yacc input in `acmecpcxml_parser.y` and the Lex input in `acmecpcxml_lexer.l`. All the classes and parser for this project are contained in the C++ namespace `acmecpcxml`.

Using the generated custom parser is simple. Just create an instance of the `acmecpcxml::XMLImporter` class, initialize it with its `Initialize()` member and import the XML data into the schema-derived classes with the `ImportFromFile()` member. The importer exposes a base class root node of the class tree via the `GetXObject()` member. This base class is then dynamically cast back to the `acmecpc` class that contains the context of the specific XML dialect defined by the `acmecpc.dtd` schema (see Listing 1).

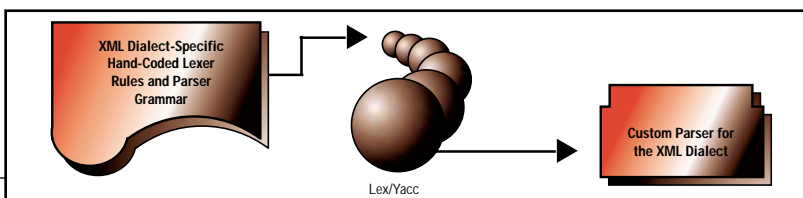


FIGURE 1 Creating a custom parser from hand-coded Lexer rules and parser grammars

AUTHOR BIO

Ken Blackwell is chief technical officer at Bristol Technology, Inc. (www.bristol.com), where he oversees product architecture and research for Bristol's eXactML (XML) product, middleware technology and transaction analysis software.

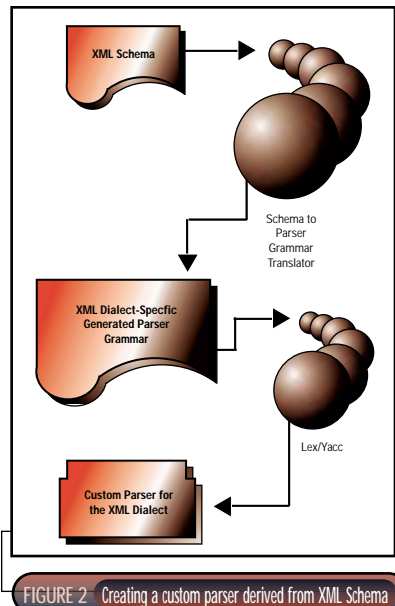


FIGURE 2 Creating a custom parser derived from XML Schema

Advantages of Custom Parser Approach

There are four primary advantages to creating a custom parser rather than using a generic parser like DOM.

1. First and foremost, it's fast. I've run benchmarks that show the custom parser to be up to three times faster than the fastest DOM parser I can find while also having a smaller in-memory footprint. The primary reason it's so much faster than DOM seems to be that it doesn't have to do dynamic validation of the XML input. Instead, validation is enforced by the automata generated by Yacc from the input files, which are derived from the XML Schema.
2. The generated parser can integrate tightly with the derived classes described in my previous article. There is no two-step process of parsing into the DOM hierarchy, then populating classes from the DOM data structures. The custom parser creates the schema-derived classes directly, without the need for the intermediate step. The generated parser can also integrate tightly with framework technologies you might be using, such as STL and MFC class libraries.
3. You get all the source code to the components that link into your application. By using the GNU-licensed Flex and Bison tools, the output source code will run on virtually every operating system imaginable. I've been very successful, for example, in running Flex and Bison on Windows NT and using the output C/C++ code on a variety of platforms with no necessary source code changes.
4. The final advantage, and the coolest of all, is that using Lex and Yacc enables you to handle those pesky XML entities more easily. I use this feature to automatically expand entities on input so my program doesn't have to worry about them. XML enti-

ties can be preprocessed just as a macro is preprocessed by a compiler when parsing a C input file. The class instances created by the custom parser contain data with entity references fully expanded. I can't stress enough the amount of headaches this little feature can save you when dealing with documents with lots of entities.

Conclusion

While XML processing may be new to the C++ community, the skills and technologies that have matured over the last decade in this community can still be very useful in handling XML data formats. In my last article I described the benefits of deriving C++ class definitions from XML Schemas. Here, I've gone a bit further to show how to derive parser grammars for XML dialects from the XML Schema.

As the XML Schema standard nears acceptance, there will be many other opportunities to reuse the work of schema designers to automatically derive programming source code, relational database schemas and other artifacts that otherwise would have to be coded by hand. C++ developers should look for these opportunities as ways to reduce the amount of repetitive work required to add or update support for specific XML dialects. ☛

KENB@BRISTOL.COM

LISTING 1

```

acmepcxml::XMLImporter importer;

// Call the acmepcxml::Initialize() function to register the
// create functions
// for the acmepcxml classes
acmepcxml::Initialize();

try {
    importer.ImportFromFile(sInputFileName, fPreprocess);
}
catch (eXactML::XException & e)
{
    std::cerr << e.GetMsg() << std::endl;
    std::cerr << "in " << e.GetSourceFile() << " at line number " <<
    e.GetSourceLine() << std::endl;
    return 1;
}

cout << "Read in XML file with no errors." << std::endl;

acmepc *acmepc = dynamic_cast<acmepcxml::acmepc *>
(importer.GetXObject());

cout << "Successfully cast XML importer root to
acmepcxml::acmepc" << std::endl;

// validate the data in the classes, throws an exception if
bad.
try {
    acmepc->IsValid();
}
  
```

```

catch (eXactML::XException & e) {
    cout << "Exception in validating XML" << std::endl;
    cout << e.GetMsg() << std::endl;
    eXactML::XMLImporterBase::DeleteImportedXObject(acmepc);
    return 1;
}

cout << "Validated acmepcxml::acmepc object" << std::endl;

// generate the XML
try {
    acmepc->EmitXML(cout);
}
catch (eXactML::XException e) {
    cout << "Exception in generating XML" << std::endl;
    cout << e.GetMsg() << std::endl;
    eXactML::XMLImporterBase::DeleteImportedXObject(acmepc);
    return 1;
}

cout << "Emitted XML to file worked fine." << std::endl;
eXactML::XMLImporterBase::DeleteImportedXObject(acmepc);
return 0;
  
```



Wireless
Spring
www.wirelessd.com

s DevCon
ead
evcon2000.com

[WRITTEN BY RICK PARFITT]

We live in the midst of a revolution. Computers have dramatically accelerated the ways we can manipulate the world and even replicate ourselves. In just 25 years growing magnitudes of computational power are making it possible to perform such human tasks as speech recognition. But without progress on many fronts, what is a machine to do with the translations of human desires? Fortunately, in this same time frame, networking and the encapsulation of data representing all aspects of our lives have reached global proportions. Computers that recognize speech can respond. While XML is a key component in making this possible, as in VoiceXML, the role goes beyond simply using new markup tags to identify data.

This article probes the technology and the capabilities of VoiceXML, a new standard adopted in May 2000 by the W3C. The initial project resulted from collaboration between IBM, Motorola, Lucent and AT&T, a story unto itself. The current list of members covers a broad spectrum of the computer industry. We will explore how VoiceXML goes beyond the graphical user interfaces of HTML and provides a framework for the most natural form of communication: spoken language.

VoiceXML can't stand alone. It's dependent on a broad set of technologies to make it useful. Let's start by reviewing the bigger picture, which we can follow with actual examples of VoiceXML.

Figure 1 is a schematic diagram of a complete system that enables a user from anywhere in the world to use the telephone and voice dialogs to interface with the growing body of data and transactions that are increasingly embodied by the World Wide Web. At the heart of Figure 1 is a VoiceXML interpreter, which operates on XML data loaded locally or from somewhere on a network. A dialog can be initiated when the Telephone Resource Manager receives an incoming call. The phone number dialed by the user, and potentially the number the user is dialing from, can be used to determine the first XML script to run. In many cases the first dialog may be used to identify the user and gather password information.

Information can be presented to the user through several channels. Currently, most telephones do not have a graphical display device and VoiceXML was not written to address the newer wireless phones that are growing in their ability to present graphical data. How VoiceXML might be integrated with other technologies such as WML is best left as the subject of another article. We focus here on the main purpose of VoiceXML: spoken language communication.

For arbitrary textual information VoiceXML can utilize Text-To-Speech (TTS). Several companies have commercially available engines that may be run as servers and be load balanced over an arbitrary number of phone lines. Every year the quality of these systems continues to improve. While the mechanical-sounding speech generated by TTS systems can still be easily distinguished from "natural" or prerecorded

VOICE
INTERACTION
FOR THE
WEB

Speech recognition applications are making major breakthroughs

speech, the intelligibility is very high. While not as pleasant as a human voice, the words are nonetheless understandable.

In many situations the information to be presented to a user can be prepared in advance. In these cases prerecorded speech may be used. In most cases a simple file server and a digital-to-analog converter system can be used to manage the prerecorded wave files. The advantage of this approach is the ability to add all the nuances of the human voice that give radio and entertainment voices their appeal. The disadvantage of prerecorded speech is the large amount of storage required and the inability to present arbitrary messages to the user. In addition, time delays may occur when attempts are made to load large wave files over a heavily loaded network.

VoiceXML also provides the ability to record and play back audio information whether from the current user or some other source, including music or other audio artifacts. These features may be used to implement voice messaging, call screening and paging services.

Information from the user comes in two primary forms: text that's been translated from spoken utterances via the automatic speech recognition server (ASR) or from decoded Touch-Tones (both Ford and General Motors are starting to combine Global Positioning with ASR to provide real-time driving directions). A number of companies currently offer very high-quality speech recognition over the telephone (see sidebar on telephone speech recognition).

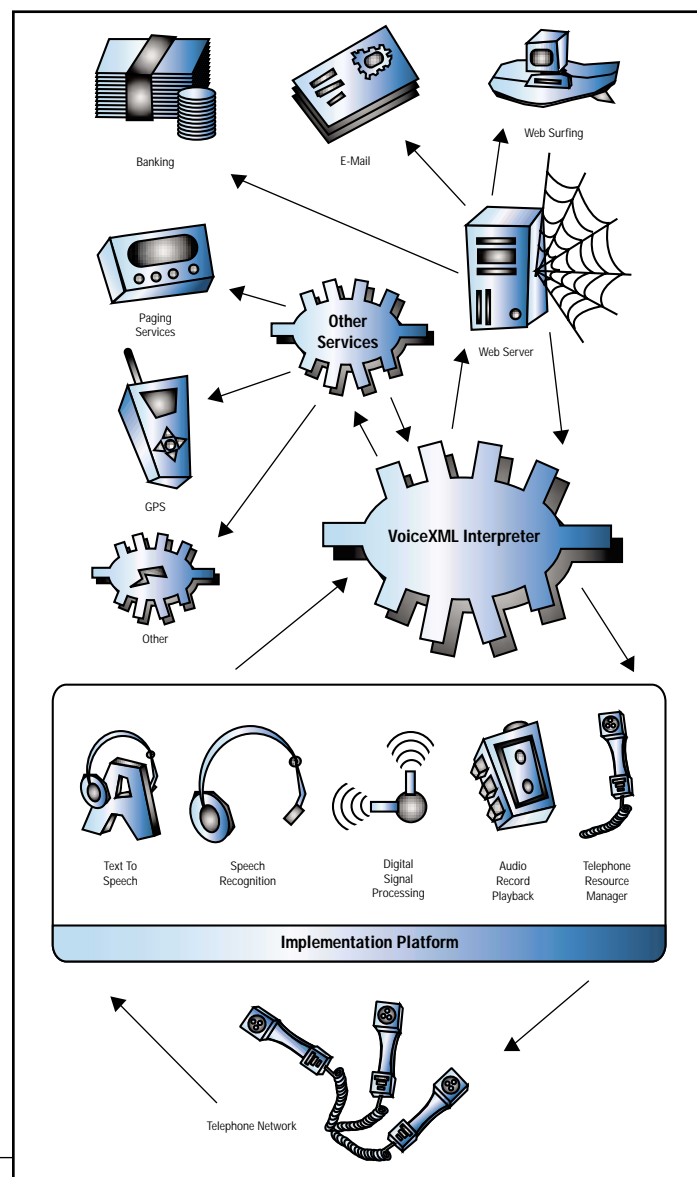


FIGURE 1 VoiceXML implementation platform

Dialogs between the computer and the user occur through a series of VoiceXML scripts that may be retrieved or dynamically generated from anywhere on the Internet. Data from legacy systems can be reformatted by a growing set of tools and won't be covered here. We'll focus on the considerations that must be made in developing Voice User Interfaces (VUI) when conducted over the telephone.

Creating Verbal Interfaces with VoiceXML

The modality of speech is substantially different from graphical interfaces. Such techniques as pop-up menus, unrestricted input fields and rapid presentation of information are not available. The old expression that a picture is worth a thousand words suggests that we need to be careful how we reveal the picture. The design examples here are simple, but give some idea of the ways in which VUIs need to be constrained.

A key value of VoiceXML, much like HTML, is the simplicity. It isn't a full-fledged programming language, and to a large extent it would be expected that, much like HTML, a new set of professionals will specialize in VoiceXML. Writing good VoiceXML dialogs requires a sense of what makes human dialog work. Some of the more proficient writers of VoiceXML dialogs that I've worked with have come from a background in linguistics, audio engineering or the broadcast industry. While XML can be a little alien for someone who has never programmed, the commands are easily learned. The art of good dialog design is an open challenge, a call to those with an understanding of verbal discourse.

We won't cover all the features of VoiceXML here. The full specification can be downloaded at www.3c.voxml.com/. There are two major constructs for building a dialog in VoiceXML: menus and forms. Menus are more restricted and essentially a subset of forms. We will focus on forms for our discussion. A form can contain various elements that we utilize to

present prompts to the user, gather input and handle various error conditions. Then, based on the user's actions, elements of the form may be used to determine which subdialog to execute next.

A form can contain several different elements that are executed in a precise order. The details of this order are fully described by the Form Interpretation Algorithm included in the VoiceXML 1.0 spec. Listing 1 shows the general flow.

We'll start with our main document, or dialog: *top_menu.vxml*. When the telephone resource manager answers the incoming call, this is the first script that will be executed by the VoiceXML processor. In the form *top_menu*, the first element, *block*, is used to play a prompt. Blocks typically contain items that are executed one time, such as introductory prompts and the initialization of variables.

The prompt tag we use here is embellished with another tag, *audio*, to reference an external prerecorded message. The data format of the audio recording and how it's played out over the telephone is platform dependent. During prototyping, or when the audio file doesn't exist, the dialog designer may specify text that will be played using the TTS system. As we have done here, placing the information inline helps to make the script self-documenting.

The form *top_menu* consists of a single field that will process the user's menu selection item. While this example is simple, and could be implemented using the menu tag, it demonstrates some of the logical operators of VoiceXML and we'll build on it in our next example. The field contains an inline grammar. The recognition engine uses the grammar to limit the acoustical search space. Simple grammar such as this works well with today's recognizers, which have been highly tuned for use over the telephone and over a wide range of noisy environments.

After a successful recognition occurs, the filled element is executed. The result of recognition is returned in the variable *main_menu*. Here we use the result to select the next script to execute.

VoiceXML uses a simple event model that passes events to a catch element. A catch element can contain executable code and can be scoped at the document, dialog or form level. The catch closest to the current scope receives the event to process. Using the throw element, new events may be defined. VoiceXML contains shorthand for several built-in events, "help," "noinput," "nomatch" and "error." Events can be used in a number of creative ways to enhance the user experience, which we will demonstrate further in our next example. Here, we use help and noinput to guide the user. The VoiceXML spec requires a built-in grammar scoped at the document level that recognizes the commands help and cancel. At any point a user may request help. The recognition platform is also required to monitor the audio input channel, throwing an event, *noinput*, when a predefined amount of time has elapsed without detecting speech. This is typically about 10 seconds and may be set as a system parameter. The recognizer is required to throw another event, *nomatch*, when the audio level at the user handset is high enough to trigger recognition but does not produce a recognizable phrase.

Messages customized to this dialog for help, noinput and nomatch are found inside the corresponding catch elements. The system also adds together the number of times the noinput and nomatch events occur and will automatically hang up when a specifiable count is reached.

As a second example, in Listing 2 we use the dialog contained in the file *weather.vxml*. We arrive here when the user says WEATHER (see Listing 1). We now need to determine which city and state to use for finding the weather. Listing 2 borrows from the early VoiceXML documentation and is an example of Java Speech Grammar Format (JSGF), with extensions to allow the return of multiple items. The W3C group is still working on the grammar specification for VoiceXML to find a general solution for representing more complex grammar that meets the needs of systems used by several companies.

This time, instead of using an inline grammar, we use an external grammar, which allows the user to specify the city or state together or separately; the order in which they are said is not important. This simple capability allows us to build more natural interfaces.

In this form we utilize a new element, *initial*. Upon entry to the form, the grammar *city_state* is activated. An affirmation prompt, "OK, let's get

SPEECH RECOGNITION TECHNOLOGY

The strides in Automatic Speech Recognition (ASR) over the past 10 years have put into place the foundation that makes VoiceXML possible. The use of better acoustic models, large amounts of training data, and statistical methods for automatically extracting the critical information from the training data have allowed speech recognition systems to work well in a number of situations. Commercial systems used with VoiceXML come pretrained. They are speaker independent. This is a necessity for use with short transactions from an arbitrary set of users over the telephone.

Another major breakthrough has been the widespread use of statistically based search algorithms for finding word boundaries. Today's systems work with continuous speech and the user is not required to leave unnatural pauses between words or phrases that older recognition systems required. Again, as we attempt to accommodate any user of the telephone, the system must be as natural to use as possible.

Finally, many of the same techniques used for recognizing speech have been successfully applied to the problems of background noise. The telephone is a particularly noisy device and today's systems do surprisingly well even when cell phones are used in a moving car.

During the last five years we have seen the proliferation of desktop dictation systems from IBM and Dragon Systems (the latter company is now part of Lernout & Hauspie). While these systems have proved useful for many people under appropriate situations, it has become clear that they are not yet a ubiquitous replacement for keyboard input. To some extent it can be argued that we're still trying to find applications where speech recognition can be universally applied.

In recent years ASR has made large inroads into automated telephone systems. One of the first, and highly successful, applications was the simple "Yes/No" system developed by Bell Labs for automating the acceptance of collect telephone calls. Most people aren't aware that a computer runs the system, and it has saved the phone companies hundreds of millions of dollars in reduced labor cost. Today literally thousands of automated phone-tree systems exist, delivering everything from movie show times to airline ticket purchases. Many of these systems have already incorporated ASR. The current systems often lack good design. The public has come to expect a low level of performance from these phone trees, which can often result in lost business and revenue. These types of automated systems are bound to improve through the technologies enabled by VoiceXML and the expanded access to information offered by the Internet.

the weather. For what city and state?" is played. This helps indicate to the user that the recognizer interpreted the last command as a request for the weather while also including a request to the user to say the city and state. We include two levels of noinput prompts. This is accomplished by the attribute count inside of the noinput event. Each time the noinput event occurs, the count is incremented by one. This allows the designer to use variety and/or offer additional information as appropriate. After the second noinput, we set the variable start to true, which causes control to leave the initial element and switch to a more directed form of input. The next field, state, has a prompt that simply asks the user to say the state for which the weather is desired. When the user says the name of a state, this information is used in the next field, city, as part of the prompt, "Please say the city in <value expr="state"/> for which you want the weather." Alternatively, the user may decide to leave the weather feature by saying "Sixty minutes," causing control to return to the dialog in top_menu.vxml.

When the dialog fills the field for city, the user is presented with an affirmation, "I'll get the weather conditions for <value expr="city"/>,<value expr="state"/>." Control is then transferred to a servlet along with the data for city and state. It'll be up to the servlet to translate the weather information for this city and state into the next VoiceXML script that executes.

GRAMMAR

Several factors affect the accuracy of speech recognition systems. The quantity and quality of the data used for training the recognizer and how well the system adapts to noise are large determinants of accuracy. Today's best speech recognition vendors are very competitive. They add to and take advantage of the latest research, constantly improving their systems. Given comparable tasks, using a given platform as intended, the performance between platforms is often comparable. Still, even today's best systems are limited in their recognition ability. When applied to simple tasks of recognizing "YES/NO", the accuracy can be close to 100%. When using the letters of the alphabet to spell over the telephone, the acoustically similar sounds of "b & v" are hopelessly confused. The art of writing good VUI designs requires an understanding of how recognition grammar works and how it affects recognition accuracy.

Several rules of thumb are helpful. Grammar made up of a short list of acoustically distinct words will generally result in good recognition. Even a long list of phrases, up to several thousand items, can result in good recognition when the items are acoustically distinct. For example, the names of companies listed on the major stock exchanges can work as grammar. Grammar that is more complex – "Two large pepperoni pizzas and one medium-size mushroom pizza" – presents several considerations. At each branch in the grammar, there are a number of choices, the branching factor. Complex grammar with many large branching factors tends to result in lower recognition, longer search times and more "out-of-vocabulary" utterances. When grammar is complex, the user may be given the false impression that it's okay to resort to "natural" English, which often results in out-of-vocabulary utterances. With complex grammar the user may also have greater trouble remembering what commands are valid, such as: "First I want two pepperoni pizzas, no, make that three, and one mushroom pizza, a medium mushroom pizza." While humans are very good at processing spoken language and extracting the meaning out of an expression that may contain many false starts and errors, it must be specified precisely in the recognition grammar or misinterpretations are likely to result. Unlike dictation systems, today's telephone recognition systems need to translate the utterances into their semantic content and produce a resulting set of actions.

Good grammar must use words and phrases that are familiar to the target user and that accurately reflect the tasks that need to be performed while offering as much flexibility and naturalness as possible.

Recognition is also affected by how the grammar is translated into acoustic patterns that are used during recognition. The text representation from the grammar is first translated using phonetic dictionaries to build the acoustic models used during recognition. Often these dictionaries need to be hand-tuned based on alternative pronunciations, and coarticulation that occurs as a result of how the sound of a word is affected by surrounding words, such as: "I want a large pepperoni pizza" or "I wanna large pepperoni pizza." Some grammar representations allow the specification of alternative pronunciations within the grammar. These fine-tunings are necessary to build highly accurate recognition systems and often require the help of a linguist or phonetician.

Getting Started

The world of VoiceXML is changing weekly. One of the first companies to offer a system for experimentation was IBM, through their alphaWorks program. They integrated an early version of VoiceXML with their ViaVoice speech technology. Most of the software can be downloaded free from www.alphaworks.ibm.com/tech/voicexml. The system supports Microsoft Windows and desktop recognition. Several companies have already deployed VoiceXML systems for their internal development work, but do not generally make their platforms available to developers. Nuance and SpeechWorks, both providers of telephone-based recognizers, have VoiceXML initiatives underway with extensive developer programs. Nuance has announced a dial-up phone system for testing scripts. It was scheduled for release in late August as of this writing and may be used free for 60 days.

[Tellme.com](http://www.tellme.com) appears to be the furthest along with their Tellme Labs developer program. They offer a free dial-up number and a range of tools for developing and testing VoiceXML scripts. They even include a window that may be used while browsing their developer site to write and modify VoiceXML scripts that can then be immediately tested from their free dial-up number.

A system can also be built by assembling the necessary components outlined in Figure 1. This approach requires a much more extensive effort and understanding in putting the parts together. Telephony cards, DSPs and various servers must all be made to work together. The most widely used recognizers from AT&T, IBM, L&H/Dragon, Nuance and SpeechWorks were not initially designed to work with VoiceXML. Several of the dynamic aspects of VoiceXML make it more difficult to simply match the respective speech APIs to the VoiceXML requirements. The VoiceXML committee is still working on a standard capable of supporting the more complex grammar that each of the recognizers referenced above are built on. This is resulting in several different VoiceXML platform-dependent solutions that are likely to change over time.

How to Find Out More

1. *VoiceXML Specification*: www.w3.org/TR/2000/NOTE-voicexml-20000505/
2. *IBM – VoiceXML software that can be downloaded and used for free*: www.alphaworks.ibm.com/tech/voicexml
3. *L&H/Dragon – Supplier of recognition and TTS systems*: www.dragon-sys.com/
4. *Nuance – VoiceXML developer tools and hosting service free for 60 days*: www.nuance.com
5. *SpeechWorks – will make VoiceXML available through their open source program*: www.speechworks.com
6. *Tellme.com – free hosting and extensive VoiceXML tools for developers and free browser for accessing a set of services (toll free dial-up number)*: www.studio.tellme.com
7. *NetByTel – offers hosting services for a fee and developer assistance*: www.netbytel.com
8. *Quack.com – offers a free browser for accessing a set of services (toll free dial-up number)*: www.quack.com
9. *General Magic – development and hosting services including VoiceXML and other tools*: www.generalmagic.com

Future of VoiceXML

Speech recognition has been under development since early work at Bell Labs in the 1950s. Progress continues to be made. The interest in speech technology comes in waves, and with each new wave new levels of performance are achieved. The current fascination with speech is compounded by the proliferation of the Internet and the ability to access it through the telephone and other wireless devices. The limits of speech recognition when used over the telephone are significant. How people will adapt to these limitations is undetermined. We do know that the widespread use of phone trees is here to stay. When VoiceXML is applied to these sorts of applications, improvements and increased acceptance are assured.

Tellme.com and Quack.com are just two of the many companies that are applying speech recognition to the more general delivery of infor-

mation and transactions currently performed by desktop computers and human agents. Most information on the Web today is not easily translated into spoken language format. We are more likely to see solutions that can be limited in bandwidth, but that take advantage of data that is becoming increasingly available through the Internet.

The transition into this new world will be made easier when a standard format for grammar representation is made by the VoiceXML working group. The efforts of the contributing companies to work together successfully to bring us the 1.0 specification of VoiceXML are to be highly applauded. The future of speech recognition has never been brighter. 🌐

AUTHOR BIO

Dr. Rick Parfitt has worked in the area of speech recognition for the past 22 years. He completed his PhD in this field through the University of California and Carnegie Mellon. During the mid-'90s he was responsible for commercializing Apple Computer's speech recognition technology. More recently he implemented two versions of VoiceXML that are used in large-scale production environments. A consultant to a number of companies that are exploring the use of VoiceXML for e-commerce and other automated information services, Rick is currently exploring the uses of WML and other wireless interfaces with VoiceXML.

PARFITT@VERIO.COM

LISTING 1

```

Main document (top_menu.vxml)
<?xml version = "1.0"? >
<vxml version="1.0">

  <form name = "top_menu" >
    <block>
      <prompt>
        <audio src = "Welcome_1.wav" >
          Welcome to 60 seconds, your complete source of news, weather, electronic commerce and personal assistant.
        </audio>
      </prompt>
    </block>

    <field name = "main_menu">
      <grammar>
        news {news}
        | get the news {news}
        | weather {weather}
        | commerce {commerce}
        | electronic commerce {commerce}
        | assistant {assistant}
        | personal assistant {assistant}
      </grammar>
      <filled>
        <if cond="main_menu == 'news'">
          <goto next = http://myXMLServer/news.vxml />
        <elseif cond="main_menu== 'weather'">
          <goto next = http://myXMLServer/weather.vxml />
        <elseif cond="main_menu== 'commerce'">
          <goto next = http://myXMLServer/ecomm.vxml />
        <elseif cond="main_menu== 'assistant'">
          <goto next = http://myXMLServer/assistant.vxml />
        </if>
      </filled>
    </field>
    <noinput>
      Please say one of the following. NEWS, WEATHER, ELECTRONIC COMMERCE, PERSONAL ASSISTANT.
    </noinput>

    <nomatch>
      I'm sorry, I did not understand you. Say NEWS, WEATHER, ELECTRONIC COMMERCE, PERSONAL ASSISTANT.
    </nomatch>

    <help>
      You have reached your 60 second access to the world of NEWS, WEATHER, ELECTRONIC COMMERCE, and your very own PERSONAL ASSISTANT. You may use simple voice commands.
    </help>
  </form>
</vxml>

```

```

    </help>
  </form>
</voicexml>
</vxml>

```

LISTING 2

```

Getting the Weather (weather.vxml)
<?xml version = "1.0"? >
<vxml version="1.0">

  <form name = "weather" >

    <grammar src="city_state.gram" type = "application/xjsgf"/>

    <initial name="start">
      <prompt>
        Ok, let's get the weather. For what city and state?
      </prompt>

      <help>
        To hear the weather, say a CITY and STATE, or say SIXTY-SECONDS to browse one of our other selections.
      </help>
      <!--If the user does not respond after two timeouts of waiting for speech we will simplify the request and only ask for city, the first field item. -->
      <noinput count="1">
        To hear the weather for any location in the US, please say a CITY and STATE.
      </noinput>
      <noinput count="2">
        <prompt>
          I did not hear anything, get weather information, please say a city and state.
        </prompt>
        <assign name="start" expr="true"/>
      </noinput>
    </initial>

    <field name="state">
      <prompt>What state?</prompt>
      <help>
        Please speak the state for which you want the weather, or say Go Back to Sixty Seconds
      </help>
    </field>
  </form>
</vxml>

```

```

    to browse one of the other selections.
</help>
</field>

<field name="city">
  <prompt>Please say the city in <value
    expr="state"/> for which you want the
    weather.
  </prompt>
  <help>
    Please speak the city for which you
    want the weather.</help>
  <filled>
    <prompt>
      I'll get the weather conditions for
      <value expr = "city" >, <value expr="state"/>
    </prompt>
    <submit next=http://ourWeatherServer/servlet/playWeather/>
  </filled>
</field>

<field name="sixtyseconds">
  <filled>
    <prompt>
      Let's try another selection.
    </prompt>
    <goto next = http://myXMLServer/ top_menu.vxml />
  </filled>
</field>
<!--If the city and state are wrong, the
user may say CANCEL>
<cancel>
  <prompt>
    Ok, Let's try again. Please say the city and state.
  </prompt>
  <clear namelist="city state start" />
</cancel>
</form>
</voicexml>
</vxml>

```

An External Grammar, city_state.gram

```

grammar city_state;

public <cityState> =
  <city> {this.city=$} [<state> {this.state=$}] |
  <state> {this.state=$}[<city>{this.state=$}] |
  <sixtyseconds> {sixtyseconds}

<city> = Los Gatos | Sunnyvale | Mountain View | San Jose |
San Francisco | Los Angeles | New York |
Buffalo | Rochester | Miami | Chicago ;

<state>= New York | California | Florida | Illinois ;

<sixtyseconds>=sixty seconds | cancel | stop ;

```



Hit

Software

www.hit.com

AvanSoft

www.avan-soft.com

FileMaker Developer 5

by FileMaker Inc.

—[REVIEWED BY JIM MILBERY]—



AUTHOR BIO

Jim Milbery is a software consultant with Kuromaku Partners LLC (www.kuromaku.com), based in Easton, Pennsylvania. He has over 16 years of experience in application development and relational databases.



FileMaker Developer 5
FileMaker Inc.
5201 Patrick Henry Drive
Santa Clara, CA 95054-1171
Phone: 408.987.7000
www.filemaker.com

Test Environment:
Gateway GPI
196MB RAM
30GB disk drive
Windows 2000

FileMaker Inc. is a venerable provider of database solutions for the workgroup environment. Once known as Claris, they've successfully shipped over 5 million units of their flagship FileMaker product around the world. On the eve of their annual user conference (in Palm Desert, California), I had the chance to work with FileMaker Developer 5 – the latest version of FileMaker's product line.

FileMaker Developer 5

FileMaker comes in four different versions: FileMaker Pro 5, FileMaker Pro 5 Unlimited, FileMaker Server 5 and FileMaker Developer 5. The core technology is contained in the FileMaker Pro 5 product, which is a workgroup database as well as an application development tool for Windows and Mac OS. Although they've passed the \$100-million mark and consider themselves an enterprise database player, I wouldn't consider FileMaker to be an enterprise-class database engine. However, they have a large number of satisfied customers who have built workgroup database applications within enterprise organizations. FileMaker itself supports ODBC access to third-party databases, and also sports a JDBC driver for the FileMaker engine. Thus it's possible to access FileMaker using a third-party tool and to use the FileMaker front end with an enterprise database such as Oracle 8i. Given the tight integration between the tool and the database engine, I'd expect that most users would use the FileMaker Pro product as a single database/tool solution. Out of the box, only 10 end users can access a standard FileMaker Pro 5 application; this is where FileMaker Pro 5 Unlimited, FileMaker Server 5 and FileMaker Developer 5 come into the picture. These three server editions allow you to deploy your applications to a larger number of end users. File-

Maker Server provides client/server-style access, while FileMaker Pro 5 Unlimited allows you to deploy applications across an intranet or via the Internet, as shown in Figure 1.

FileMaker Developer 5, the tool I worked with, assists you with the process of deploying your FileMaker applications

First Impressions

FileMaker Developer 5 includes FileMaker Pro 5 and the FileMaker Developer 5 extensions for deploying applications. You can download both products from the FileMaker home page or you can install the product from the two-CD set. The starting point for developing applications is the FileMaker Pro 5 interface. Existing FileMaker programmers will notice that FileMaker has added some new features to the developer interface with this release. The IDE has been redesigned to be consistent with the Microsoft Office look and feel. This makes sense, since FileMaker is focused on the workgroup environment. There's support for additional dialog box types, field formats (including 3D) and some additional report formats that are accessible through the new layout assistant. FileMaker's ODBC support has been enhanced to make it easier than ever to access

FileMaker databases from third-party tools such as Visual Basic – or to access enterprise data from within FileMaker applications. For Windows developers FileMaker 5 adds support for ActiveX automation, which enables developers to control multiple programs from a FileMaker script. One advantage of FileMaker 5 is that it's been designed for the needs of workgroup programmers. Thus it's relatively easy to get started with the product. The installation includes a number of sample databases as well as a tutorial, but I chose to import my existing Net University database into FileMaker. It's a simple process to import a database into FileMaker and then use the layout assistant to build a report or form as shown in Figure 2.

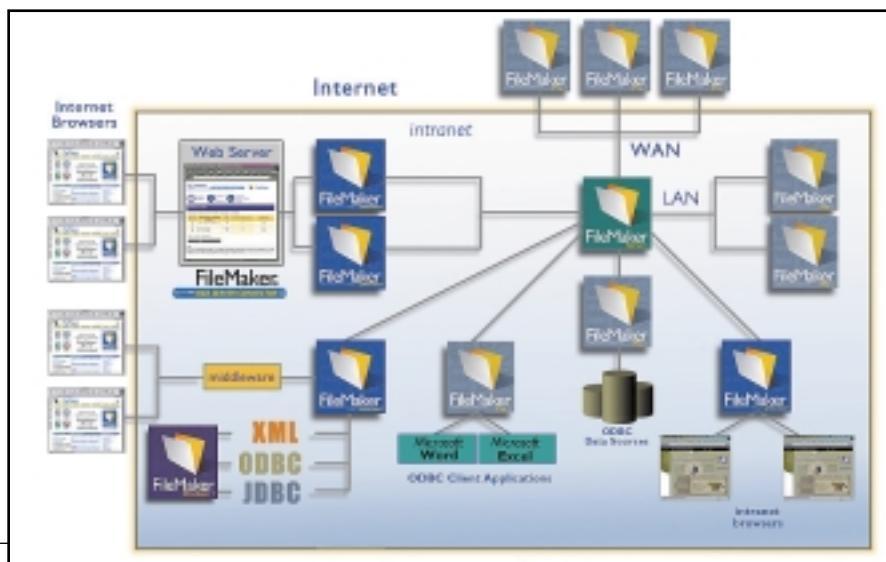


FIGURE 1 FileMaker development and deployment architecture

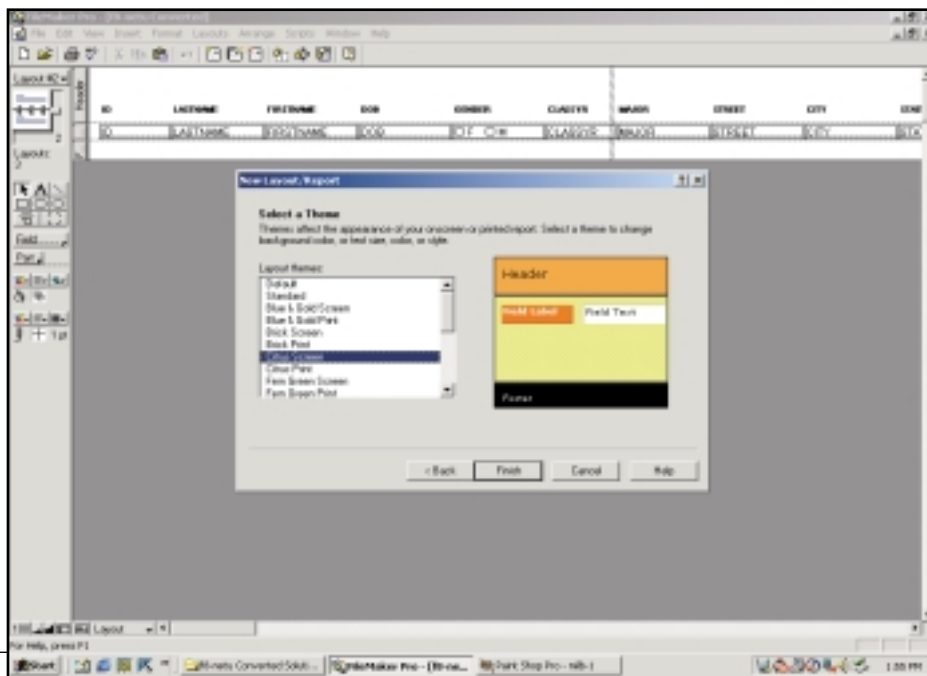


FIGURE 2 FileMaker layout assistant and IDE

FileMaker provides all the standard capabilities of an application development environment including forms and report painters, field definition panels and a scripting language. The advantage of using a tool like this is that it automates many of the routine programming tasks that developers face, such as building query forms and formatting data. Proprietary scripting languages generally provide a higher level of abstraction from the data (as compared to coding directly in Java, for example), but the price for this increased productivity is that you're locked in to a single vendor. Nevertheless, I found FileMaker made short work of the process of creating reports and forms – an important consideration for workgroups that are short on programming resources. Within a few minutes I had several working forms that included query-by-example functions.

FileMaker, The Web and XML

When it's time to deploy your FileMaker applications, you have a number of choices. FileMaker's support for both Windows and Mac OS is somewhat unique for a proprietary tool. This makes the product appealing for mixed-platform workgroups within a corporation. Furthermore, the Web deployment options allow you to create an application that could be hosted on a commercial ISP site (provided you can live with the Windows/Mac operating system restriction). FileMaker Pro 5 allows you to export your applications as a series of static HTML files or to deploy the application as a client/server program. You can also deploy the application as a dynamic Web application using a proprietary Web tag technique called CDML (Claris Dynamic Markup Language). Via CDML, FileMaker commands are embedded into standard HTML files that are then processed by a server engine. (Allaire's ColdFusion works the same way.) FileMaker provides their

own Web server for this purpose (FileMaker Web Companion), or you can use their CGI interface or Web Server Connector to connect to a third-party Web server, such as Apache, with your FileMaker code. Small departments will probably appreciate the simplicity of using the Web Companion, but it doesn't provide all the features of a Microsoft IIS or iPlanet WebServer platform. From the brief look I had at the FileMaker forums on the FileMaker home page, I got the impression that the documentation and resources for the Web Server Connector are a little weak. FileMaker's Instant Web Publishing feature doesn't require a working knowledge of CDML, but you'll need to work with

CDML if you want to customize the output – this is really where FileMaker Developer comes in. Through the Developer interface you can create kiosk-style applications and work directly with CDML. I'm not sure how scalable the Web Companion would be for a large application, but it was certainly easy to work with. It was a simple matter of setting some preferences in my database to make the NetU application available through a browser (see Figure 3).

The FileMaker Web Companion also allows you to output XML from your FileMaker databases. There's a white paper on their Web site (www.filemaker.com/downloads/pdf/xml_overview.pdf) that provides an excellent overview of both XML and FileMaker's XML capabilities. FileMaker positions XML as a means for offloading work from the server to the client (thus increasing scalability), which I don't quite agree with. Nevertheless, the white paper provides a clear, readable explanation of using XML and FileMaker together. I'd expect that most users would be using FileMaker's XML capabilities to publish data to the Web (as opposed to using XML as a data integration technology). Toward this end, FileMaker includes a number of XML grammars to assist you in the process of formatting data and some clear examples that walk you through the process of applying stylesheets to your XML data.

Summary

FileMaker provides a solid product for workgroup database computing and Web publishing, but it doesn't replace the need for an Oracle 8i or DB2. The development environment and scripting language are easy to master, albeit proprietary, and the Web publishing features are easy to use. Programmers new to XML may find that FileMaker provides a simple environment for building dynamic XML-based applications without the learning curve of an enterprise-class tool set.

ID	LASTNAME	FIRSTNAME	DOB	GENDER	CURRENT	HIRING	STREET	CITY	STATE	ZIP	PHONE
1	SMITH	John	12/15/1977	M	Y	1999	1234 Main St.	Springfield	IL	60001	312-555-1234
2	SMITH	Matthew	10/25/1977	M	Y	1999	5678 Elm St.	Springfield	IL	60002	312-555-5678
3	SMITH	Phillip	10/25/1977	M	Y	1999	9101 Main St.	Springfield	IL	60003	312-555-9101
4	SMITH	Julia	02/15/1977	F	Y	1999	1234 Main St.	Springfield	IL	60004	312-555-1234
5	SMITH	Robert	02/15/1977	M	Y	1999	5678 Main St.	Springfield	IL	60005	312-555-5678
6	SMITH	Matthew	10/25/1977	M	Y	1999	9101 Main St.	Springfield	IL	60006	312-555-9101
7	SMITH	Matthew	10/25/1977	M	Y	1999	1234 Main St.	Springfield	IL	60007	312-555-1234
8	SMITH	Matthew	02/15/1977	M	Y	1999	5678 Main St.	Springfield	IL	60008	312-555-5678
9	SMITH	Jane	02/15/1977	F	Y	1999	9101 Main St.	Springfield	IL	60009	312-555-9101
10	SMITH	Matthew	10/25/1977	M	Y	1999	1234 Main St.	Springfield	IL	60010	312-555-1234
11	SMITH	Robert	02/15/1977	M	Y	1999	5678 Main St.	Springfield	IL	60011	312-555-5678
12	SMITH	Matthew	10/25/1977	M	Y	1999	9101 Main St.	Springfield	IL	60012	312-555-9101

FIGURE 3 NetU database using FileMaker Web Companion

JDJ Store
www.jdjstore.com

Internet World

www.pentonevents.com

[WRITTEN BY JOSHUA FOX]

X

ML is often used to transmit messages. The tags indicate where the message should go and how it should be handled. The information transferred in the message can be XML as well.

Just as modularity is necessary in coding programs, separating the messaging envelope from the data body is necessary in planning data structures. Keeping the layers from recognizing each other allows developers to work separately, and to change implementations when necessary. Allowing the layers to interact too much is a common mistake, which leads to complicated and inflexible code.

In this article I'll show you how to build an effective XML-based layered message architecture.

Header/Body Design Pattern

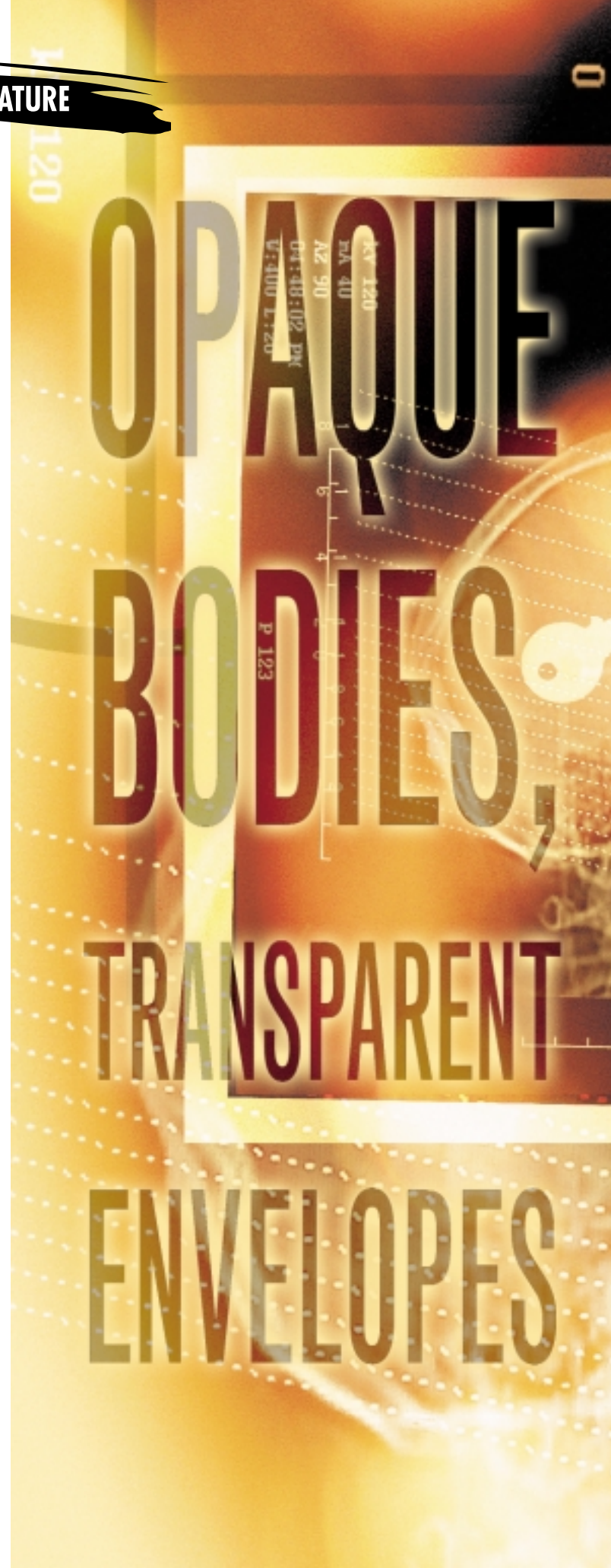
The "Header/Body" design pattern is the most effective way to place a document into a message envelope without creating undue dependencies between data and messages. The "envelope" is the structure of the message, and includes body and header. The "body," also known as the "payload," is the information to be sent, recognized by the higher level of abstraction in the code, while the "header" is additional information added by the lower envelope level for its own purposes, such as routing. (see Figure 1) It's essential that these layers be independent.

Opaqueness and transparency are two sides of the same coin. *Opaqueness* means that the lower layer envelope knows nothing about the body. *Transparency* means that the higher-layer body knows nothing about the lower layer over which it is sent. The transparent lower layer might be sent over an even lower layer, and then the same principle arises: the transmitted data should be opaque. The benefits of opaqueness and transparency are complementary: opaqueness allows you to switch body formats at will, while transparency allows you to switch message formats. Where opaqueness and transparency aren't preserved, the layers become dependent on each other, and change becomes difficult.

The Header/Body design pattern is a data-structure pattern, appropriate for XML. As such, it differs conceptually from the better known behavioral patterns, which are appropriate for object-oriented languages such as C++, Java or Smalltalk. Yet data-structure and behavioral patterns do have a lot in common. The Header/Body data-structure pattern shares a "Layers" pattern language with behavioral patterns such as Bridge, Adapter and Façade as these keep levels of abstraction separate and minimize mutual dependencies. In real life the data-structure and behavioral patterns are closely related: when data structures are carefully encapsulated in the appropriate layers, the processing code can be encapsulated as well. And since violation of layer encapsulation is as much a problem in data structures as in object behavior, I'll explain in this article the challenges involved in designing the layers and keeping them separate.

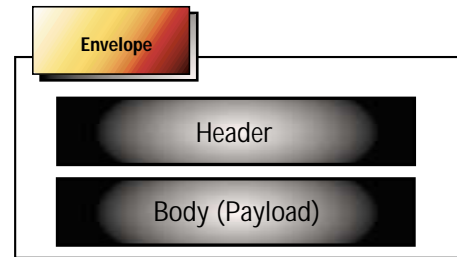
Layered Architecture

To understand the Header/Body pattern, we divide the participants along two axes: sender and recipient, higher and lower layers (see Figure 2). The sender on the higher layer wishes to send a message in a given XML Schema. It packs this body in an envelope by adding the header,



Check out the design benefit of multilayered Header/Body structures

Basic Structure for Layered Message



Basic XML Layout for Layered Message

```
<Envelope>  
<Header> ... message header ... </Header>  
<Body> ... message body ... </Body>  
</Envelope>
```

FIGURE 1 Structure of layered XML message

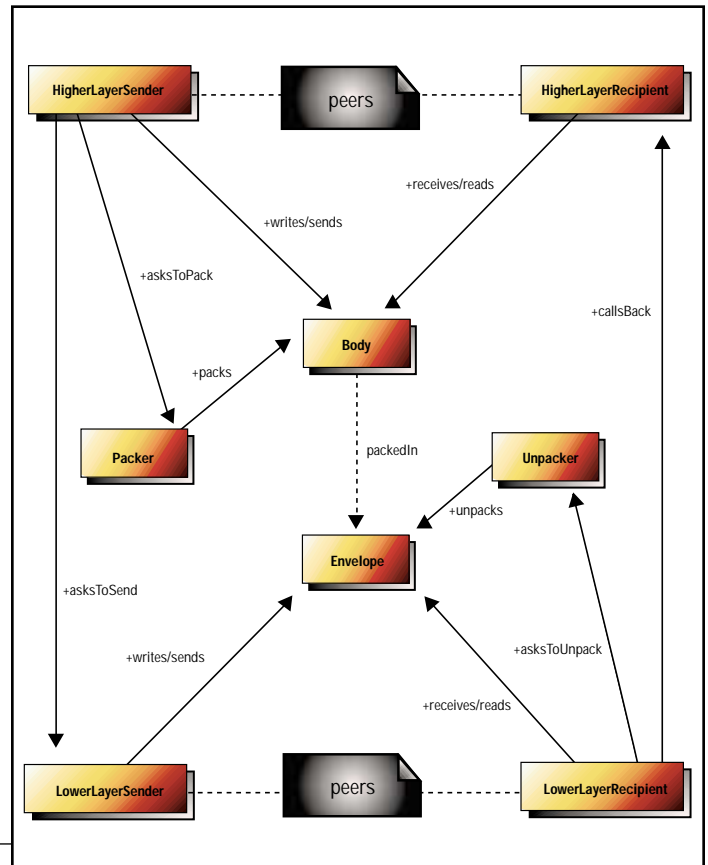


FIGURE 2 Participants in layered message system

perhaps using an auxiliary packer class. The higher-layer sender then passes the envelope, with body packed inside, to the lower-layer sender, which transmits it over the communications protocol (itself an even lower layer). The lower-layer recipient receives the message, unpacks the body, perhaps using an auxiliary unpacker class, and then passes the body to the upper layer in a callback function (see Figure 2).

In actual systems, protocols are stacked many layers high, and the message itself is the body of a lower-layer envelope. XML is layered on top of non-XML layers. For example, SOAP (Simple Object Access Protocol) defines XML headers and bodies in an XML envelope. SOAP is sent

over various lower-layer transports such as HTTP or asynchronous messaging. Information on processing the method-call body is extracted into the SOAP envelope. From there the namespace identifying the method may actually appear one layer up, in a special SOAPAction HTTP header, as this example, abridged from the SOAP proposal, shows.

```
POST /StockQuote HTTP/1.1
...
SOAPAction: "some-URI"
...
<SOAP-ENV:Envelope ... >
  <SOAP-ENV:Body>
    <m:GetLastTradePriceDetailed xmlns:m="some-URI" >
  ...
```

If HTTP-parsing software such as a CGI had to access the SOAP headers or even the body directly, HTTP and XML instructions would be hopelessly mixed.

Layers in Enterprise Software

The complexity of typical distributed software makes it imperative to separate layers. Taking as an example a typical enterprise system (see Figure 3), a servlet receives a POST from an online form, then creates an XML purchase order using an industry-standard schema. The servlet passes the XML to an EJB, which decides on the message's destination. A Java Message Service queuing system, also used by other applications like payroll and billing, has a standard XML schema generally designed for routing messages in the enterprise. The EJB wraps the purchase-order-schema XML in the messaging-schema XML, attaching destination information, then passes the XML to software that reads the messaging XML and interprets it for routing through the JMS system to the order fulfillment server. Here an order fulfillment application uses an EJB, which applies the fulfillment business logic and accesses data in a database using the database's propriety XML-to-RDBMS system.

Various data formats are used: two XML schemas, along with non-XML formats like forms, Java remote method calls, JMS messages and a relational database.

To control this complexity, we must isolate the applications and their data formats from each other. For example, if we switch databases, we'll need a different XML-to-RDBMS layer, and if we switch from JMS to CORBA, we need to rewrite the XML-to-messaging code.

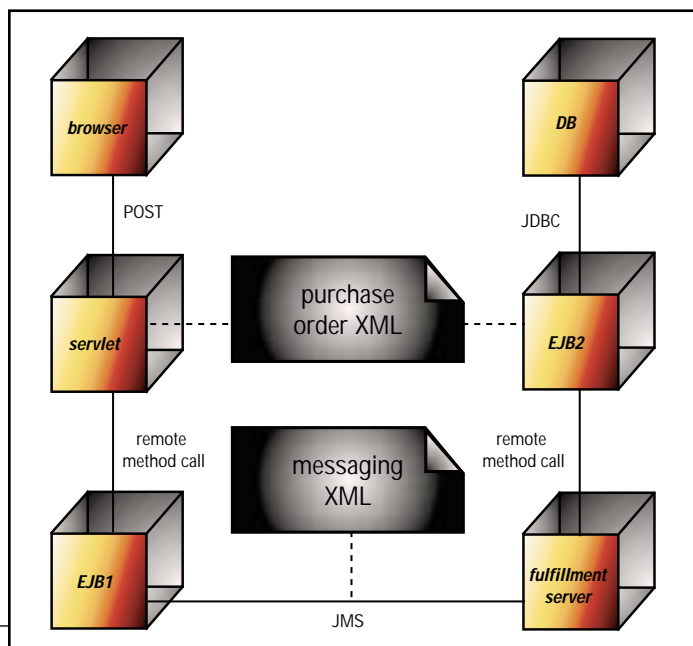


FIGURE 3 Example of distributed enterprise software

The separation of layers doesn't prevent all coupling. There must be coupling between peer components. Senders and recipients on the same level of abstraction must understand the same schema. The identification of the schema or DTD in the document itself can provide a degree of flexibility in parsing, but to process the document meaningfully, the software on each side must have similar abilities. This means that peers on a given layer are tightly coupled.

Transformation vs Packaging

Packaging bodies in messages is a way of passing data of one schema to software that requires a different schema. Transforming a schema, as with XSLT, is another. Packaging and transformation have different purposes. Packaging is used for passing data between layers of abstraction, where each layer has a clearly defined functionality that must be preserved. Transformation, on the other hand, is used for passing data between participants at the same layer of abstraction, where the functionality of one ceases and the other begins. With transformation, information loss is permissible since the full functionality of the transformed schema is no longer needed. For example, an XML weather report contains application-specific tags allowing software to understand it, but when the XML is transmitted over WAP to a cell phone display, it's transformed into WML by XSLT; now all that matters is that the result be human-readable on the cell phone, and machine-readable semantics can be erased.

Interlayer Communication

When the lower layer envelope needs information about the higher layer body, but the body must be kept opaque from the moment of transmission, you can apply the Header/Body pattern.

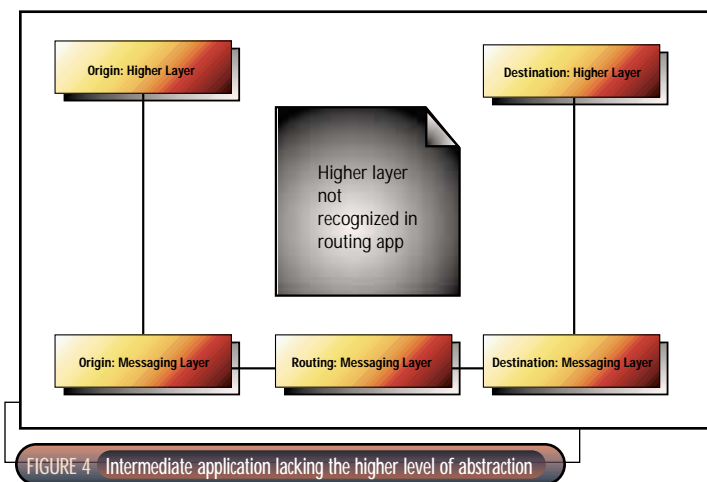
In XML the Header/Body pattern is implemented with a schema that assigns elements for the header and the body. SOAP provides a good illustration of this format. The SOAP Message is an XML document with a root Envelope element. This Envelope has zero or more header elements as its first children, with one or more body elements as the headers' siblings.

When the body is passed from the higher layer to the lower layer, it is "packaged" into the envelope and the header is added. Data needed by the lower layer is extracted from the body and placed in header fields; other header fields may be added at this time. Only the class charged with packing the data need know the formats of both body and message headers, as it's essential that lower layers do not extract these body elements en route. Typical header data includes:

- **Routing information:** The intended destination(s) of the message and whether it should be routed through requested intermediate points. This helps the lower layer send the data on its way without examining the content of the body.
- **Priority:** How urgent the message is.
- **Body type:** Whether the body is, for example, a JPEG image, an MP3 sound file, a Java serialized object or XML. MIME types are commonly used for this purpose. Subtypes are possible: if the body is XML, the schema URL and processing instructions could be included here.
- **Pricing information:** If the messaging layer includes a mechanism for buying and selling data, how much the data in the body costs, and where payment should be sent.
- **Batching:** Whether this message should be taken as part of a batch of messages that have something in common.
- **Transaction:** Whether this message is part of a data-access transaction, with the identifier for the transaction.
- **Authentication:** The source of this message, with a public-key authentication header to prove it.
- **Logging:** Recording various fields to a log file.

Intermediary Applications Missing a Layer

You must be careful about layer violation when an intermediary in transmission has to intervene on the higher level of abstraction. For example, say that a user interface application builds a purchase-order XML document from information taken from a GUI, then puts that document as the payload of an XML messaging envelope and sends it to a routing application. The routing app examines the message headers and



envelope. The parser must in effect pass over the body twice: once to get the whole body, and again to extract from the body fields that are appropriate for the envelope header. In this implementation both passes are made at once, as two XML documents are simultaneously accumulated. Processing instructions are removed from the body, as is the initial `<?xml ...?>` directive. This is necessary to keep processing instructions in the body from being wrongly applied to the envelope layer. The parsing may also remove comments. Note that the XML Infoset specifications say it's legal to remove comments along the way in processing XML; this also implies that you can't use comments to "escape" things that could confuse the envelope layer, such as processing instructions.

SCHEMA WITH A FEW CONSTRAINTS

The body may be required to have a simple schema, to assure that it has basic information the header needs. For example, the body might be required to note the destination of transmission so the appropriate header can be populated (see Listings 1, 2 and 3 for MessageSchema.xml, which has a Body with a TargetAddress element).

SUBCLASS OF SCHEMA

You can keep the benefits of a body schema, yet gain more freedom by extending the schema. In XML Schema this is done with the base and derivedBy attributes. For example, you may want to extend the Body element to allow information to be sold over the messaging system. Just add some pricing information to the information already given by the base Body element.

```
<simpleType name="PricedBody" base="Body" derivedBy="extension">
  <element type="Price"/><!-- defined elsewhere-->
</simpleType>
```

UNPARSEABLE DATA

If you want the data to be a completely opaque character string to the envelope, you can wrap it in a CDATA section. Anything wrapped by `<![CDATA[` and `]]>` is ignored by XML parsers. This way, any XML parsers on lower layers won't waste time parsing and validating data. Complete opaqueness requires that the lower layers completely ignore structures in the higher layers. A validation error in the body shouldn't interest developers of the messaging system; that's the responsibility of sender and recipient on the higher layer. For example, if the lower layer doesn't have access to schemas from the upper layer, it can't validate the XML transmitted in message bodies. CDATA blocks have the additional advantage of saving parsing time, since lower-level DOM message parsers will automatically parse the body XML, even though the lower layer can't make use of the parsed information.

Another way to convert parseable XML into unparseable character data is to "escape" the entities in the XML body. A call to the DOM method `envelope.createTextNode(body)` (envelope is a Document and body is a String) can achieve this effect. In the process all less-than signs ("`<`") are converted to `<`; greater-than signs ("`>`") are converted to `>`; ampersands ("`&`") are converted to `&`. This produces a text node that isn't parseable XML but just a character data string as far as the parser is concerned.

A caveat: neither the CDATA nor the entity-conversion techniques can be used recursively. If your message XML document has a CDATA section, and you try to embed this message as the body of a message at an even lower layer, then you have one CDATA section nested in another. The standard doesn't permit this, and a parser will start a CDATA section at the first `<![CDATA[`, ignore the nested `<![CDATA[`, then end the CDATA section at the first interior closing symbol `]]>`. This isn't what you want (see Listing 4).

Likewise, if a message includes `<` and `>`; entities converted from "`<`" and "`>`" symbols in a body, and that message itself undergoes conversion when packed in yet another message, then the `<` and `>`; entities from the two conversions will be mixed up. When you try to resolve the entities to less-than and greater-than signs, as part of the unpacking process, there's no way to know which entities underwent packing once or twice, and need unpacking respectively once – to the intermediate message layer – or twice – to the body layer.

decides where the message goes – in this case to the order fulfillment application, which not only fulfills the order, but also builds XML to report back its success to the user interface application. The UI then converts this XML with XSLT to HTML, appropriate for the user interface. In principle it seems that the routing application should know nothing about how the UI and the order fulfillment app talk with each other: the higher layer is missing in the routing app (see Figure 4).

But what if the routing app can't figure out where to send the message, or can't contact the order fulfillment application? The routing app needs to send XML back to the UI indicating failure, and this XML has to be in the higher-layer format, appropriate for conversion into user-readable information. A common mistake is to construct the user response within the intermediate application, which means using code inappropriate for this application. A more correct architecture defines a schema for reporting faults on the messaging layer back to the originating application, where user-interface XML can be more appropriately constructed.

How to Implement

Ideally, with body-layer data in hand, you should be able to send it over any envelope layer without change. Sometimes, though, the application places constraints on the body, and then transparency is reduced (see Table 1). In this section I'll show some examples of why you might want to impose transparency- or opaqueness-reducing constraints.

Degree of Constraint	Constraints	Example Below
more	Body has a schema defined by lower layer that poses extensive constraints on body content.	SOAP
	Body has a schema which imposes few constraints beyond the root element.	MessageSchema.xml with schema-validating parser
	Body has a schema which is derived from a root schema that imposes few constraints.	Schema PricedBody
	Body is XML with no schema used.	MessageSchema.xml when schema is ignored
	Body is unparseable character data.	CDATA or "escaped" XML
less	Body is unparseable binary data.	Base64

TABLE 1 Degree of constraint on body

STRUCTURED BODY

If you specify the structure of the body, you can be sure to get information from the body that may be necessary in processing it. For example, the "body" in a SOAP header is application-specific. It isn't an arbitrary XML document; rather, it must fit specifications for XML that are appropriate for a method call, return value or fault report.

The example Packager/Unpackager, available online, shows the basic outlines of how to work with XML-in-XML. It uses SAX and Java to package a body in an envelope, simultaneously extracting header information from the body and inserting it into the appropriate place in the

ENVELOPE AND BODY IN INCOMPATIBLE FORMATS

You achieve even more opaqueness when the body is of a completely distinct format from the message envelope. In that case the envelope is completely incapable of understanding the body.

Given two distinct formats like XML and Java objects, how do you embed one opaquely in the other? It's easy to embed XML in a Java object, since XML is just a string and a string is a Java object. You achieve full opaqueness when you set a JMS StringMessage, since the string can contain anything, not just a well-formed XML document. Likewise, you can send XML as a parameter in an RMI method call, which, behind the scenes, relies on object serialization.

Conversely, you might want to send Java objects over the wire. Java objects provide many of the advantages of XML: self-describing data, automatic metadata discovery (dynamic class-loading, analogous to automatic downloading of schemas from URLs) and platform independence. Of course, Java objects are closely tied to the Java language, creating an unwanted dependency but providing the ability to download behavior along with data. Several serialization methods have been proposed for Java and other languages through XML: SOAP, WDDX and SOX are a few – but ordinary Java object serialization has the advantage of being easily available in a robust implementation. You may also find Java serialization in legacy systems.

A Java object embedded in XML serves as an illustration of how to achieve complete opaqueness in XML. Here the data isn't even a meaningful string, but simply a sequence of raw bytes. The techniques used to encode a Java object can be used to opaquely encode any binary data. While there are a number of ways of doing this, the most common is to encode the data in Base64. This standard uses the characters from a-z, A-Z, 0-9, + and / to represent the digits of a Base64 number, each digit encoding 6 bits. Code for Base64 conversion is commonly available, since Base64 is used to encode binary data in HTTP (Web) and SMTP (e-mail) transmissions.

Just using Base64 data doesn't let the parser or XML application know that the string is encoded Base64 information. By creating an additional constraint, you indicate that Base64 is the encoding scheme used. In W3 XML Schema the base type binary with an encoding schema component containing an attribute value="base64" indicates that the element is of type binary. A further refinement (with concomitant reduction of transparency) is to indicate that the Base64 data is specifically a Java object. You derive your own data type from binary, declaring that the element is specifically a serialized Java object. If this schema is used, only serialized Java objects should be passed as the element's character data.

```
<simpleType name="serializedJavaObject" base="binary">
  <encoding value="base64"/>
</simpleType>
```

A Base64 encoding string looks like nonsense to a human reader. A simple Java object comes out looking quite opaque: r00ABXNyAAZQZXJzb27QohB9ajN37IAAUkABG1BZ2V4cAAAACc=. The value of this shouldn't be underestimated, since breaking the layer structure is an all-too-common human error. When you don't allow other developers access to your data format – even if it isn't actually encrypted – you reduce the temptation for coders of the lower layer to read fields from the upper layer. While this shouldn't necessarily make you choose human-unreadable data – indeed, the XML philosophy supports human-readable data – enforcing the discipline of layer separation should be a top priority.

The examples given above show that the lower layer often imposes requirements on the body, breaking transparency and opaqueness. There is a way around this. If it's impossible to rework the body with the constraints imposed by the lower layer, you can always treat the constraints as headers in yet another layer of envelope, and place the body in that. For example, you can't pass arbitrary XML in a SOAP body, but you can define a SOAP method call that takes arbitrary XML as a parameter, then pass the XML through that method.

GLOSSARY

Body: The lower-layer information packed into the appropriate part of the envelope structure

Constraints (on body): Requirements imposed by one layer on another

Coupling: Knowledge held by one software component about another, so that neither can be easily changed

Envelope: The structure within the message; includes Header and Body (see Figure 1).

Escaping: Converting characters that are significant to the data structure on a given layer to characters that aren't structurally significant

Header: Information in the body needed by the lower layer; brought into the header from the body during packing

Higher layer of abstraction: Logically closer to the user

Lower layer of abstraction: Logically closer to the machine

Message: The unit of data that holds the envelope structure (see Figure 1)

Opaqueness: Lack of information about the higher layer in the lower layer

Packing: Inserting the body and header information into the appropriate parts of the envelope structure

Transparency: Lack of information about the lower layer in the higher layer

Header/Body: A Non-XML Example

The Header/Body pattern isn't new to XML; it's commonly found in many communication protocols, of which the IP protocol stack may be the most familiar. Here each layer, from the Application Layer down through the Host-to-Host, Internetwork and Network Access layers, treats data passed down to it as raw binary information, ignoring any application-specific formatting, flow control, routing or checksum information. A higher layer informs the lower layer of any information it needs through APIs. For example, an application-layer HTTP client passes down the server's address through the Host-to-Host socket API, and certainly doesn't expect the socket to parse the HTTP text stream. Conversely, the lower layers inform the upper layers of needed information through return values and interrupts. No information is exchanged between layers though the raw byte streams.

Each layer prepends its own header to the packet of higher-layer raw data. This simple linear arrangement, which is the origin of the "Header/Body" terminology, contrasts with the nested arrangement of XML, which has the advantage of self-describing structure. Thus an XML message envelope can have multiple variable-length headers and bodies, with their borders neatly delineated by the XML tags.

Efficiency vs Encapsulation

Developers have a tendency to break both opaqueness and transparency. Too often the lower-layer software reads body data, whether for efficiency or expedience.

The usual excuse is efficiency; efficiency and encapsulation are often at odds. The problem has come up in implementations of the IP protocol stack. When lower layers were made to depend on the format of higher layers for the sake of efficiency or error recovery, protocol layers have been locked together so that changes in layers have become impossible.

The same dilemma can occur in XML. Envelopes within envelopes, each one assigned to a separate layer, can cause a tremendous overhead, and you might be tempted to save some of the tags within tags.

Nevertheless, the XML philosophy comes down firmly on the side of good design. The XML 1.0 Spec includes a list of 10 design goals. After worthy goals such as straightforwardness and compatibility, the end of the list is occupied by "Terseness in XML markup is of minimal importance." This seems counterintuitive, but the XML philosophy consciously rejects space efficiency. This philosophy would value encapsulation over efficiency, and in fact encapsulation can solve the problem of efficiency. By applying compression techniques that take advantage of XML's redundancy, bulkily wrapped XML can be compressed to net-worthy sizes. And even if a price must be paid in unzipping and unwrapping XML envelopes, the design benefit of multilayered Header/Body structures is considerable. ☺

References

1. *Design Patterns*: <http://hillside.net/patterns/>
2. *Object serialization in XML*:
with Web Distributed Data eXchange – www.wddx.org
with Simple Object Access Protocol– www.w3.org/TR/SOAP/
3. *Defining data types with Schemas*: www.w3.org/TR/xmlschema-2/
4. *Java Message Service*: www.javasoft.com/jms
5. *XML Infoset*: www.w3.org/TR/xml-infoset
6. *Base64*: www.ietf.org/rfc/rfc2045.txt

7. *XML Spec*: www.w3.org/TR/REC-xml

AUTHOR BIO

Joshua Fox is senior architect at Surf&Call Network Services, where he develops distributed Java applications. His current project is an Internet service that enables joint Web-surfing. Joshua has a BA in mathematics from Brandeis University and a PhD in comparative Semitic philology from Harvard.

JTFOX@USA.NET

LISTING 1

```
<?xml version="1.0"?>

<Message xmlns="x-schema:MessageSchema.xml">
  <Header>
    <!-- When this document was built, routing
    information was copied from the body to the
    header, to avoid breaking opaqueness during
    transmission -->
    <TargetAddress>
      joe@anywhere.com
    </TargetAddress>
  </Header>

  <!-- This is an opaque body. It could
  include text, markup elements, or both.
  Schema for item placed in the body
  should be defined separately, and available
  to sender and recipient in a
  common repository. Because it is not CDATA,
  this body is not completely opaque: It will
  be parsed and must be well formed and valid if
  the parser requires that.-->

  <Body>
    <Order xmlns="x-schema:OrderSchema.xml">
      <Target>
        joe@anywhere.com
      </Target>
      <Item>
        Garden chair
      </Item>
    </Order>
  </Body>

  <Body>
    <!-- The second example of a body element of
    this message is totally opaque. It demonstrates
    the embedding of an XML message in another
    where the body's schema is not currently
    available. This data will be left unparsed
    (saving parsing time) until the recipient
    "unpacks" it. It does not even have to be
    well-formed or valid.-->
    <![CDATA[
      <?xml version="1.0"?>
      <Purchase xmlns="x-schema:
http://www.unavailable.com/purchaseSchema.xml">
        <Price currency="USD">54.00</Price>
        <SKU>209238</SKU>
      </Purchase>
    ]]>
  </Body>
</Message>
```

LISTING 2

<!--
To preserve opaqueness of body, this Schema is defined separately from any schema that may be placed in the body, such as the OrderSchema.

This example uses Microsoft's XML-Data pre-standard variant of Schemas, for which validators are currently available. Standard Schemas should be quite similar when the recommendation is approved by the W3. Some examples in the text are based on that proposal, since Microsoft XML-Data does not

```
support derivation.
-->
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">

  <ElementType name="TargetAddress" content="textOnly"/>

  <ElementType name="Header" content="eltOnly">
    <element type="TargetAddress"/>
  </ElementType>

  <!-- Note that the content of the body is left
  open, to allow for opaqueness. -->
  <ElementType name="Body" content="mixed" model="open"/>

  <ElementType name="Message" content="eltOnly">
    <element maxOccurs="1" minOccurs="1" type="Header"/>
  <!--This schema allows for multi-part bodies
  with maxOccurs="*" -->
    <element maxOccurs="*" minOccurs="1" type="Body"/>
  </ElementType>
</Schema>
```

LISTING 3

<!--
To preserve transparency of messaging format this Schema is defined separately from any schema that may be placed in the body, such as the MessageSchema.

-->

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="Target" dt:type="string" content="
textOnly"/>
  <ElementType name="Item" dt:type="string" content="
textOnly"/>

  <ElementType name="Order" content="eltOnly" model="open"
order ="seq">
    <element type="Target"/>
    <element type="Item"/>
  </ElementType>
</Schema>
```

LISTING 4

<!--This is a counter-example! Don't try this yourself! The CDATA section, as it would be incorrectly understood by a parser, is highlighted. -->

```
<outerMessage>
  <![CDATA[
    <intermediateMessage>
      <![CDATA[
        <innermostBody/>
      ]]>
    </intermediateMessage>
  ]]>
</outerMessage>
```



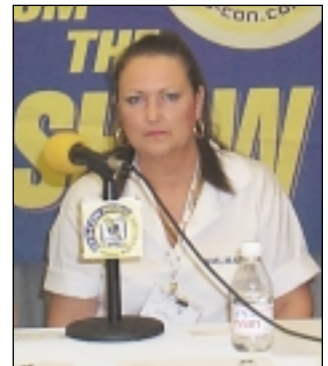


Interview...

with **SANDRA CLARK**

CHIEF EXECUTIVE OFFICER INSIGHT TECHNOLOGIES

AN INTERVIEW BY AJIT SAGAR



XML-J: Can you give me an idea of how you got started and what the background is? Did the developer start off with XML or did he or she actually have some other technologies in place?

Clark: No, in fact XMLMATE, the product we launched at XML DevCon 2000, originated after the development of one of our other products. That product, Intellego, is a full-blown application that automates the structuring and indexing of XML.

XML-J: The basic technology your company started out with was XML?

Clark: No, COM/DCOM. A lot of our work was in COM/DCOM. We were struggling with some of the learning curves of XML. That's really how we developed XMLMATE – we needed rapid development tools to assist us internally. A lot of products originate this way; you develop them for a problem you've got. And we realized that there are a lot of developers out there who would want a similar tool.

XML-J: Are you going to continue in the other technology areas or are you going to concentrate on XML?

Clark: We'll be in several areas, certainly COM/DCOM. We're in XML now, and we're looking at Java/CORBA as well.

XML-J: So what are you doing with COM/DCOM?

Clark: We have a full-blown application, Intellego suite, that is an intelligent rules

engine with support for conversion from various file formats. Using artificial intelligence, we can extract not only the inherent metadata from multiple file formats, but we can actually create additional metadata packaging by using artificial intelligence and applying business rules. We can then encapsulate an object and maybe attach a digital signature to it, route it into anything from a document management system or reference management system to a B2B- or B2C-type portal. We can export out, having created the metadata in any format such as XML, PDF or HTML. You name it.

XML-J: So you also do the PDF?

Clark: Yes, we do PDF as well.

XML-J: Let's talk about XML technology, what you've started out with and where you plan to go in the long term.

Clark: Certainly. XML is the emerging standard from our perspective and we feel we're in a very fortunate position to be at the start rather than the me-too environment. Certainly XMLMATE is an excellent order converter, a very good intelligent editor that allows rapid development for experienced developers, but it also helps shorten the learning cycle for those who are less experienced with XML. We present data in easily understandable forms, such as a table with full procedures, so it

gives somebody a chance to see XML code and to utilize that and get up and running very fast.

There are interesting things like the XML binary object storage of any binary information, for example, images and a database inside of an XML document.

XML-J: Are you using the PC data, the XML data tech or are you doing something more than that?

Clark: We're doing something more than that. I'm not deeply technical. You can tie me up in knots pretty fast.

XML-J: I was just wondering because XML does support a binary format that it can use to transfer data, and from what I've heard of their performance I was wondering if Insight had done anything in that direction?

Clark: The performance for the transfer of binary object data within an XML object is very fast using XMLMATE, both composing and decomposing. You can send the binary format with an XML object anywhere, and you can restore it at the other end.

XML-J: This is in the XML editing area, right?

Clark: Yes, XMLMATE is a powerful XML editor, but it's more than XML editing. The focus in XML editing is the structuring of the data. A lot of non-XML data formats can be well presented via XML. XMLMATE is an ideal tool for this

research/development. We provide various types of conversion from non-XML into XML with an advanced editing facility. XMLMATE gives developers a quick start as an XML analysis tool. You can send an HTTP request to a Web server that returns XML, then copy the reply to the clipboard and paste it into XMLMATE for analysis of your ASP/CGI codes. You can run SQL query, get results into XML, view the structure and get your own ideas about how you would like this to be done. Or build an XML Document Type Definition (DTD) and use it as a template for a blank XML document. XMLMATE as an XML editor helps shorten the development cycle. It did for us. We had the same problems as everyone going to XML.

XML-J: Okay, the flagship product you have is XMLMATE?

Clark: Yes. XMLMATE is the flagship product in our XML range. It's \$149 for the professional version so it's geared toward the individual developer, but we also have corporate pricing. We're looking very much to the corporate server market.

XML-J: I was trying to lead to that. As an editor myself it will be hard to sell products. I was wondering who you are partnering with or planning to partner with?

Clark: Our whole strategy as a company is to produce products suitable for OEM and as toolkits. First are XMLMATE and Intellego, these types of products. We started with Intellego and are in serious discussions with several major software corporations regarding this product. We've been approached for XMLMATE as well because obviously people are doing lots of things like file conversions, and they want a tool that they can OEM into their product.

XML-J: I can see you being approached by some of the XML server vendors and the apps server ven-



XML is the emerging standard from our perspective and we feel we're in a very fortunate position to be at the start



XML DevCon

www.xmldevcon.com

dors, even as acquisition targets in the event you have something they could include in their own tool offering.

Clark: Absolutely, and that's our aim. Trying to be a direct sales company is not the way to go in my view. Obviously, people will download it off the Web, but we are targeting the OEM area of the market also.

XML-J: Are you making any headway there?

Clark: I think it's too early to name any names. However, we're very pleased with the quality of companies that we're talking to. Obviously, we will be very grateful for any extra help we can get from anybody. It's all about networking and the show actually was an excellent forum for that.

XML-J: Yes, I think so.

Clark: We certainly are going to do some advertising as well.

XML-J: That's why we're trying to do two or three interviews a month so that vendors can get some exposure through the magazine.

Clark: It's certainly very appreciated. We're very excited. When you launch a new product you're never sure how well you'll be received, but we had several hundred people actually have demonstrations and the response from those people was excellent, so we were very pleased.

XML-J: I received an e-mail with a link about www.XMLBoutique.com. How did that name come about?

Clark: We were looking for a good domain name. I don't need to tell you how hard it is to find a dot-com name that will indicate your subject matter, and we didn't want to limit it just to XMLMATE. We feel XML Boutique is a Web site where other XML products would be available. And that is the intention – a boutique, a store. That's really how the name came about. So we see XMLMATE as the first tool set.

XML-J: I understand this is an editing and conversion tool. Do you see this

spreading into overall Enterprise applications, to B2B applications?

Clark: In B2B applications?

XML-J: Someone like WebMatters, for example, because they have the adapters to extract data from ERP resources. Where would you see yourselves playing in that area or, for that matter, eXcelon or Orion? Some of the XML server vendors that specialize in extracting the data as opposed to creating, you are creating the tools but obviously that's not the area in which other vendors are concentrating, to be able to process that data. Where do you see your product fitting in?

Clark: Certainly Intellego is probably well suited for that because that's what it does. It takes any data source and parses the data through several times. It has an inference engine and uses artificial intelligence, and then you can route back the extracted data into any application. It's excellent for taking any form of data and applying more intelligence to it, then passing it through to another application or directly into a Web publishing system, or whatever you want to do. Our system is different.

The strength is in the fact that Intellego utilizes a declarative rules engine that allows more than one solution as the output of a business rule. The declarative nature of the engine enables it to read, load and apply rules to incoming data, generate new rules or conclusions based upon processed information, take necessary actions and output the result into external applications. When we looked at the marketplace, there are people who are doing niche parts of what Intellego does, like e-mail extraction.

Companies out there are looking at the vertical market, such as the legal market. We've looked at it as a big picture. Again, as with most development, it usually results from a customer requirement. However, we built the product because we have a lot of experience in software development, and there's nothing worse than developing a product around a custom application and trying to "productize" it as an afterthought.

We built it from the ground up with the knowledge that we would then take this product to market. And that is very, very key.

XML-J: Tell me about this rules engine now. What kind of capability does it have and how do you see it being used in a business application?

Clark: Intellego Suite is an intelligent rules engine with built-in support for various data conversions. It was based upon a logical model of information feeders and we consider it an intelligent information filter that enables the supply of filtered information to the search engines and record management/document management systems. The artificial intelligence lives in the rule processor, which is based upon Prolog language and language extensions, developed by Insight. One of the language extensions is XMLProlog, which allows the handling of structured data in an efficient way on a symbolic level. XMLMATE was built as a tool to help with XML data processing inside XMLProlog.

The Intellego rules engine has built-in functionality to parse various data types into XML and output results into a variety of file formats, including XML. Most organizations are having problems with applications like document management, record management and knowledge management systems. They've got portals, but what they don't have is the ability to create the metadata automatically and to apply intelligence to that.

XML-J: You know that most of the vendors that actually have more products, they usually have that on their tool and others so how do you feel this is distinguished from these tools?

Clark: I think it's the fact that from our research we developed a piece of technology that's a missing link that other vendors don't have and we've been looking at some very complex applications that are starting to come through. As people become more knowledgeable, applications and requirements get more complex; for example, the longevity of a record is now a key issue. We

can encapsulate a record with metadata to create an object with business rules attached to it. The object encapsulated may contain thousands of documents. We can then attach a digital signature to it for long-term preservation of the record, independent from any application. This would be a typical application. Intellego takes data from a variety of feeds, so you could have a news feed coming in and you could have legacy database information, Lotus Notes. It's across the whole basis, as opposed to people who just specialize in Web documents or Microsoft Exchange.

So we're looking at a much bigger picture. And that, we believe, is our strength and certainly, from all the research we've done, nobody is doing quite what we are doing. This is what makes it very attractive.

XML-J: About standards. You said Insight was also thinking of moving into that world. Do you know what kind of migration you plan to do there?

Clark: We've built our software to adhere to industry standards in such a way that it will enable us to move the base architecture into other environments. From my perspective in Insight, I'm making sure we're going to support the most important environments. Therefore, the whole Microsoft COM/DCOM is one thing, but a lot of people out there are going the Java/CORBA route. Therefore we want to be able to offer our applications for that environment also.

XML-J: When do you plan to do that? What is the road map?

Clark: Time scale. We'll be looking at Java/CORBA within the next couple of months. We're probably looking to release the CORBA version of Intellego toward the end of the year and further versions of XMLMATE over the next few months. ☛

Ajit Sagar is the founding editor and editor-in-chief of XML-Journal.



AJIT@SYS-CON.COM

“We’re looking at a much bigger picture. And that, we believe, is our strength... nobody is doing quite what we are doing”

ADVERTISER INDEX

ADVERTISER	URL	PH	PG
AIR2WEB	WWW.AIR2WEB.COM		27
AVANTSOFT	WWW.AVANTSOFT.COM		51
CAPE CLEAR	WWW.CAPECLEAR.COM	353-1-241-9900	15
CAREER OPPORTUNITIES			65
HIT SOFTWARE	WWW.HIT.COM		51
IBM	WWW.IBM.COM/DEVELOPERWORKS	800-772-2227	68
ICON INFORMATION SYSTEMS	WWW.XMLSPY.COM		4
INFOSHARK	WWW.INFOHARK.COM	888-DATASHARK	9
INSIGHT	WWW.XMLBOUTIQUE.COM		11
INTERNET WORLD	WWW.PENTONEVENTS.COM	800-500-1959	55
IXIASOFT	WWW.IXIASOFT.COM		19
JDJ STORE	WWW.JDJSTORE.COM	888-303-JAVA	54
NETDIVE	WWW.NETDIVE.COM		33
PROGRESS	WWW.SONICMQ.COM/AD11.HTM	800-989-3773 EXT. 450	3
SOFTQUAD SOFTWARE	WWW.SOFTQUAD.COM/PRODUCTS/XDMETAL/EVAL/	800-387-2777	2
SYS-CON MEDIA, INC.	WWW.SYS-CON.COM	800-513-7111	32
THE SHORT LIST	WWW.THESHORTLIST.COM		17, 65
XML DEVCON FALL 2000	WWW.XMLDEVCON2000.COM	212-251-0006	63
XML GLOBAL TECHNOLOGIES	WWW.GOXML.COM	206-352-8334	67
WIRELESS DEVCON	WWW.WIRELESSDEVCON2000.COM	212-251-0006	44, 45



OUR BUSINESS. Etensity is a leading eBusiness professional services firm helping companies capitalize on the enormous opportunities of the Internet. The digital marketplace is an historic opportunity for business inception. Etensity cultivates results with an inclusive service approach featuring strategic consulting, creative design, technology deployment, maintenance, and security services inside a strong Etensity culture.

Focus...
what separates us from everyone else.

OUR PEOPLE. Etensity is always recruiting for a variety of innovator positions and encourages resume submissions. Currently we are seeking enthusiastic, bright individuals who can fill the following roles: Analysts, Developers, Designers, as well as Project Managers with expertise in web design, development and/or analysis.

OUR CULTURE. The most important element in our business model is our Etensity culture. We've nurtured this culture through a clear company message encouraging quality, curiosity, innovation, loyalty, and fun.

Washington, DC: etensity.inprofile@humansolution.com
 New York: etensity.nyc@humansolution.com
 California: etensity.ca@humansolution.com

Visit us at www.etensity.com

e-business n-focus

TheShortList


www.theshortlist.com

XML NEWS


Bristol Technology Announces eXactML 1.2

(Danbury, CT) – Bristol Technology Inc. announces eXactML 1.2, with support for Sun Solaris SPARCompiler 4.2 and XHTML.

eXactML 1.2 XML-enables C++ applications by generating object-oriented interfaces for reading and writing valid XML content based on any DTD or schema. Using new parsing technology that proved to be at least three times faster than DOM, eXactML reduces the time and expertise needed to support XML.



An evaluation version can be downloaded from the site.  www.bristol.com/exactml/erf.htm

Synth-Bank Provides Briefing Video

(Tacoma, WA) – A video created for the benefit of decision-making executives, "Introduction to XML" explains the how and why from such definitive authorities as Charles Goldfarb, John Bozak, Tim Bray, Michael Sperberg-McQueen and Jean Paoli, the people who created the technology. The 79-minute presentation (VHS only) covers "XML in Business," "History of XML," "Theory of Markup Languages," "Introduction to XML Syntax," "XML in the Real World" and "Information Stewardship."  www.synthbank.com

AbriaSoft Partners with O'Reilly & Associates

(Fremont, CA) – AbriaSoft, a provider of database solutions, is partnering with O'Reilly & Associates, a technology information company. Abria SQL Standard, AbriaSoft's first packaged commercial  distribution of MySQL, will include *MySQL and msql*, a book written by Randy Jay Yarger et al., from O'Reilly.  www.abriasoft.com www.oreilly.com

 **O'REILLY** 


American Century Chooses Arbortext Software

(Ann Arbor, MI) – American Century Investments, a leading investment management company serving nearly 2 million indi-

Arbortext


vidual and institutional investors, has chosen Arbortext's Epic e-content software as the basis to create and produce a variety of innovative investment product offerings.  www.arbortext.com

SilverStream Announces SOAP Support in xCommerce

(Billerica, MA) – SilverStream Software, Inc., has certified xCommerce, its B2B integration server product  family, for information exchanges using the Simple Object Access Protocol (SOAP) 1.1 reference



specification. xCommerce can interoperate with any other SOAP-compliant server to receive SOAP-formatted requests, execute appropriate "behind the firewall" Web services and deliver SOAP-formatted responses.

SilverStream

SilverStream xCommerce provides visual tools for integrating XML-formatted business transactions, trading partner agreements and partner interface processes. It also provides enterprise-level B2B capabilities not defined in the SOAP reference specification, including security, directory services, connection and state management, load balancing and systems failover.  www.silverstream.com

SoftQuad Software Joins WAP Forum

(Toronto, ON) – SoftQuad Software Ltd. has joined the Wireless Application Protocol (WAP) Forum to advance the delivery of XML content to mobile users. The Forum is an industry association that has developed the de facto world 

 **SoftQuad** standard for wireless information and telephony services on digital mobile phones and other wireless terminals.  www.softquad.com www.wapforum.com



Microsoft Releases Test Version of E-Commerce Software

(Redmond, WA) – Microsoft has released its first public test version of BizTalk Server. The product is part of a family of e-commerce software, dubbed .Net Enterprise Servers, that Microsoft will soon ship, including its SQL Server 2000 database.


The test version of BizTalk Server can be downloaded from Microsoft's  Web site. www.microsoft.com 

Software AG Brings EntireX 5.3 to U.S. Market


(San Ramon, CA) – Software AG Inc. is now shipping EntireX 5.3, a message-oriented middleware for fast, mission-critical application integration. Functioning as a bridge, EntireX helps organizations define an integration architecture for enhancing existing applications, linking up legacy applications with new, standard software or adding new custom applications.

As part of the company's suite of native XML products, EntireX 5.3 features an XML wrapper that allows XML-based communication with traditional, non-XML-enabled applications such as host CICS software. Additional enhancements include integration with Tamino, a native XML server and  Software AG's hallmark XML offering, and enhanced support for CORBA.  www.softwareagusa.com

New Allora Product by HiT for XML Web Development

(San Jose, CA) – HiT Software has a new Web interoperability middleware that gives application developers high-performance XML access to relational databases. HiT Allora gives IT organizations the advantage of a consistent XML data access model, synchronous SQL transaction access, asynchronous message queue access and significantly higher performance compared to generic DOM development. 

HiT Allora

HiT Allora offers two powerful modes: catalog browsing and SQL query. In the first mode the entire DBMS catalog can be dynamically browsed to retrieve information about catalogs, schemas, tables and columns. In the second mode queries can be executed against an RDB server and result sets returned as an XML document.  www.hit.com

HiT SOFTWARE

XML Global Technologies

www.goxml.com

IBM
**[www.ibm.com/
developerworks](http://www.ibm.com/developerworks)**